

# Bias-Aware Sketches

Jiecao Chen  
Indiana University Bloomington  
jiecchen@uemail.iu.edu

Qin Zhang  
Indiana University Bloomington  
qzhangcs@indiana.edu

## Abstract

Count-Sketch [6] and Count-Median [11] are two widely used sketching algorithms for processing large-scale distributed and streaming datasets, such as finding frequent elements, computing frequency moments, performing point queries, etc. The errors of Count-Sketch and Count-Median are expressed in terms of the sum of coordinates of the input vector excluding those largest ones, or, the mass on the tail of the vector. Thus, the precondition for these algorithms to perform well is that the mass on the tail is small, which is, however, not always the case – in many real-world datasets the coordinates of the input vector have a non-zero bias, which will generate a large mass on the tail.

In this paper we propose linear sketches that are *bias-aware*. They can be used as substitutes to Count-Sketch and Count-Median, and achieve strictly better error guarantees. We also demonstrate their practicality by an extensive experimental evaluation on both real and synthetic datasets.

## 1 Introduction

Linear sketching algorithms, such as Count-Sketch [6] and Count-Median [11], are powerful tools for processing massive, distributed, and real-time datasets in a *space/communication* efficient way. Such algorithms typically consist of two phases.

1. *Sketching phase.* Let  $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$  be an input data vector. We apply a linear sketching matrix  $\Phi \in \mathbb{R}^{r \times n}$  ( $r \ll n$ ), on  $\mathbf{x}$ , getting a sketching vector  $\Phi\mathbf{x}$  whose dimension is much smaller than the input vector  $\mathbf{x}$ .
2. *Recovery phase.* Using  $\Phi\mathbf{x}$  to recover useful information about the input vector  $\mathbf{x}$ .

In the distributed model, the input vector  $\mathbf{x}$  is the coordinate-wise sum of the  $t$  vectors  $\mathbf{x}^1, \dots, \mathbf{x}^t \in \mathbb{R}^n$  distributed at  $t$  sites, which connect to a central coordinator. The goal is for the coordinator to learn  $\mathbf{x} = \sum_{i \in t} \mathbf{x}^i$  communication efficiently. Note that the naive solution that each site sending  $\mathbf{x}^i$  to the coordinator is communication expensive. By linearity we have  $\Phi\mathbf{x} = \Phi\mathbf{x}^1 + \dots + \Phi\mathbf{x}^t$ . Thus each site can simply send the local sketching vector  $\Phi\mathbf{x}^i$  to the coordinator, and then the coordinator sums up these local sketching vectors to obtain the global sketching vector  $\Phi\mathbf{x}$ , from which it reconstructs  $\mathbf{x}$  using the recovery procedure. The total communication will be  $t$  times the dimension of  $\Phi\mathbf{x}$ , which is much smaller than the dimension of input vector  $\mathbf{x}$ .

In the streaming model [2], where items arrive one by one in the online fashion, a new incoming item  $i \in [n]$  corresponds to updating the input vector  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_i$  where  $\mathbf{e}_i$  is an all-0 vector except the  $i$ -th coordinate being 1. Again due to linearity, we can easily update the linear sketch as  $\Phi\mathbf{x} \leftarrow \Phi\mathbf{x} + \Phi\mathbf{e}_i$ . The space usage of the streaming algorithm is simply the dimension of the sketch  $\Phi\mathbf{x}$ , which is again much smaller than the dimension of  $\mathbf{x}$ .

**The Basic Problem.** In this paper we consider the basic problem that in the recovery phase, we want to best reconstruct the input vector  $\mathbf{x}$  using the sketching vector  $\Phi\mathbf{x}$ . More precisely, our goal is to design a sketching matrix  $\Phi$  and a recovery procedure  $\mathcal{R}(\cdot)$  with the following properties.

- *Accuracy.* The maximum coordinate-wise difference between the recovered vector  $\hat{\mathbf{x}} = \mathcal{R}(\Phi\mathbf{x})$  and the original vector  $\mathbf{x}$  is small, or,  $\|\mathbf{x} - \hat{\mathbf{x}}\|_\infty$  is small.
- *Compactness.* The size of the sketch (equivalently,  $r$ , the number of rows of  $\Phi$ ) is small;
- *Efficiency.* We can compute  $\Phi\mathbf{x}$  and  $\hat{\mathbf{x}} = \mathcal{R}(\Phi\mathbf{x})$  time-efficiently.

**Applications.** This basic problem has numerous applications in massive data processing. Once obtaining a good approximation of  $\mathbf{x}$ , we can answer a number of queries such as *point query* and *frequent/top- $k$  items* (if the  $\ell_\infty$ -norm of  $\mathbf{x} - \hat{\mathbf{x}}$  is small), *range query* and *frequency moments* (if the  $\ell_p$ -norm of  $\mathbf{x} - \hat{\mathbf{x}}$  is small, for  $p = 1, 2, \dots$ ), etc. These queries have numerous real-world applications, including Internet data analytics [10], search engines [20], data stream mining [9], streaming and distributed query processing [7, 8, 27], etc. See [23] for an overview.

One task that existing linear sketches cannot perform well but the ones that we are going to propose excel, is to detect *outliers with small frequencies*, or, *rare items*, which is very useful for detecting abnormal events. We shall see an example shortly. We will also empirically test our proposed algorithms on outlier detection and compare their performance with existing linear sketches (Section 4).

**Background.** Let us start with some background on linear sketches. For a general vector  $\mathbf{x} \in \mathbb{R}^n$ , it is impossible to recover  $\mathbf{x}$  *exactly* from the sketching vector  $\Phi\mathbf{x}$  of a much smaller dimension. However, in many cases we are able to recover  $\mathbf{x}$  up to some small errors. One such error guarantee, called the  $\ell_\infty/\ell_p$ -guarantee, is that for any  $\mathbf{x} \in \mathbb{R}^n$ , letting  $\hat{\mathbf{x}} = \mathcal{R}(\Phi\mathbf{x})$ , the coordinate-wise error of the recovery is bounded by

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O(k^{-1/p}) \cdot \text{Err}_p^k(\mathbf{x}), \quad (1)$$

where  $k$  is a tradeoff parameter between the sketch size and the accuracy guarantee, and

$$\text{Err}_p^k(\mathbf{x}) = \min_{k\text{-sparse } \mathbf{x}'} \|\mathbf{x} - \mathbf{x}'\|_p,$$

where we say a vector is called  $k$ -sparse if it contains *at most*  $k$  non-zero coordinates. In other words,  $\text{Err}_p^k(\mathbf{x})$  is the  $\ell_p$ -norm of the vector containing all coordinates of  $\mathbf{x}$  except zero-ing out the  $k$  coordinates with the largest absolute values. We often call the  $k$  largest coordinates the *head* of  $\mathbf{x}$  and the rest  $(n - k)$  ones the *tail* of  $\mathbf{x}$ . Note that if  $\mathbf{x}$  is  $k$ -sparse, then we are able to recover it exactly since  $\text{Err}_p^k(\mathbf{x}) = 0$ .

We typically consider  $p = 1$  or  $p = 2$ , and the error guarantee in Equality (1) can be achieved with high probability by the Count-Median algorithm [11] and the Count-Sketch algorithm [6] respectively; we will illustrate these two algorithms in details in Section 2.

It is folklore that  $\ell_\infty/\ell_1$  and  $\ell_\infty/\ell_2$  guarantees can be converted into  $\ell_1/\ell_1$  and  $\ell_2/\ell_2$  guarantees respectively (see, for example, Section II of [17]). More precisely, for  $p \in \{1, 2\}$  we can derive from Inequality (1) that

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_p = O(1) \cdot \text{Err}_p^k(\mathbf{x}), \quad (2)$$

which gives a more intuitive approximation guarantee on the whole vector instead of individual coordinates.

**Data Bias and Bias-Aware Sketches.** The question we would like to address in this paper is:

*What if the coordinates in the input vector  $\mathbf{x}$  have a non-trivial bias?*

Let us consider an example. Let  $k = 2$ ,  $n = 10$ , and

$$\mathbf{x} = (3, 100, 101, \mathbf{500}, 102, 98, 97, 100, 99, 103). \quad (3)$$

We have  $\text{Err}_1^k(\mathbf{x}) = 700$ ,  $\text{Err}_2^k(\mathbf{x}) = \sqrt{69428} \approx 263.49$ , which are pretty large. It is easy to see that these large errors are due to the fact that most coordinates of  $\mathbf{x}$  are close to 100 (intuitively, the bias), which results in a heavy tail. It would be much better if we can remove this bias first and then perform the sketching and recovery.

In this paper we propose *bias-aware* sketches that achieve the following performance guarantee. Let  $\beta^{(n)}$  be the  $n$ -dimensional vector with  $\beta$  at each coordinate. For  $p \in \{1, 2\}$ , our sketches can recover an  $\hat{\mathbf{x}}$  such that

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O(k^{-1/p}) \cdot \min_\beta \text{Err}_p^k(\mathbf{x} - \beta^{(n)}). \quad (4)$$

Same as Inequality (2), for  $p \in \{1, 2\}$  we can derive from (4) that

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_p = O(1) \cdot \min_\beta \text{Err}_p^k(\mathbf{x} - \beta^{(n)}). \quad (5)$$

We define the  $\beta$  value that minimize the RHS of Inequality (4) the **bias** of the input data vector  $\mathbf{x}$ .

Clearly, the RHS of Inequality (4) is no more than the RHS of Inequality (1) (equal when the best bias  $\beta$  is 0). In the case when the all except at most  $k$  coordinates of  $\mathbf{x}$  are close to a non-zero  $\beta$ , our error bound is much better than that in (1). For the example mentioned earlier, we have  $\min_\beta \text{Err}_1^k(\mathbf{x} - \beta^{(10)}) = 12$  and  $\min_\beta \text{Err}_2^k(\mathbf{x} - \beta^{(10)}) = \sqrt{28} \approx 5.29$  ( $\arg \min_\beta = 100$ ), which are significantly smaller than those given by Count-Median and Count-Sketch.

We note that in the application of detecting top-2 outlier coordinates, for the vector  $\mathbf{x}$  in (3), Count-Median and Count-Sketch cannot detect the coordinate with value 3 since their errors are much larger than 3 or even the median 100, while the errors introduced by our bias-aware sketches are small enough for this job.

**Remark 1** Compared with the single bias  $\beta$ , one may want to allow multiple bias values. For example, for the data vector  $\mathbf{y} = (200, 100, 50, 50, 50, 50, 10, 10, 10, 10)$ , one may want to use two bias values  $\beta_1 = 50$  and  $\beta_2 = 10$ . Unfortunately, this cannot be done if we want to obtain an  $o(n)$  size sketch where  $n$  is the dimension of the input vector, simply because when we have at least two bias values, in the recovery procedure for each of the  $n$  coordinates of input vector we need the information of which bias value has been deducted from that coordinate, which costs at least 1 bit.

**The BOMP Algorithm.** We note that Yan et al. [28] studied the basic problem on *biased  $k$ -sparse* vectors (a *restrictive* class of vectors where all coordinates of  $\mathbf{x}$  are equal to some unknown value  $\beta$  except at most  $k$  “outliers”), and used it for distributed outlier detection. Besides the theoretical deficiency,<sup>1</sup> the recovery procedure, named BOMP (biased Orthogonal Matching Pursuit), in [28] is in an iterative fashion which is

<sup>1</sup>The correctness analysis for the algorithm BOMP by Yan et al. is based on two unproven conjectures, and does not extend to general vectors, while we give a *complete* theoretical analysis for recovering general vectors with coordinate-wise error bounded by the RHS of (4), and our sketch size  $O(k \log n)$  for general vectors is better than the claimed (but unproven) sketch size  $O(k^{1.1} \log n)$  for biased  $k$ -sparse vectors in [28].

very time consuming. Therefore it cannot handle large size vectors within a reasonable amount of time. See a brief description of BOMP in Section 4.1 and its practical performance Section 4.3.

**Our Contributions.** In this paper we have made the following contributions.

1. We have given a rigorously formalization of the bias-aware sketches, which *strictly generalizes* standard linear sketches in the error guarantees.
2. We have proposed bias-aware sketches with  $\ell_\infty/\ell_1$  and  $\ell_\infty/\ell_2$  error guarantees. We have also shown how to implement these bias-aware sketches in the streaming model for fast real-time query.
3. We have implemented our algorithms and verified their effectiveness on both synthetic and real-world datasets. We note that our algorithms significantly outperform the Count-Sketch algorithm and Count-Median algorithm in terms of accuracy on five basic queries listed above.

**Related Work.** The history of data sketch/summary can be traced back to Morris’ *approximate counter* [22] and Flajolet and Martin’s *probabilistic counting* algorithm [16]. Subsequently, streaming algorithms were extensively investigated since the seminal paper [2] by Alon et al. Among algorithms in the streaming model, Count-Sketch [6] (with a refined analysis by [21]) and Count-Min/Count-Median [11] were found particularly useful in a number of applications from data analytics and mining to query processing and optimizations. However, these algorithms cannot handle data bias.

Deng et al. [13] attempted to remove the bias in the Count-Min algorithm. At the time of recovering a coordinate mapped to a counter  $C$ , their algorithm averages all other counters to obtain an estimate of the bias presented in  $C$ . It turns out that such an estimation is too rough to be useful – their analysis shows that their algorithm can only achieves comparable recovery quality as Count-Sketch.

Yan et al. [28] formulated the bias recovery problem in the context of distributed outlier detection. As discussed earlier, they focused their discussion on the biased  $k$ -sparse vectors, and did not give a solid theoretical analysis. Moreover, as mentioned, our experiments have shown that their Orthogonal Matching Pursuit (OMP) based recovery algorithm is very time expensive (thus impractical) to recover vectors of large sizes.

Our work is closely related to the area of compressive sensing. In fact, our linear sketching and recovery algorithms can be seen as natural extensions of the standard compressive sensing sparse recovery algorithms [5, 12, 14] (see [4, 17] for introductions). In the standard sparse recovery setting the bias of the vector is assumed to be 0, which does work well for a number of problems in signal processing but its power is somewhat limited for massive data processing where coordinates in vectors may have non-zero biases. We note that the idea of debiasing can be viewed as a special case of the incoherent dictionary learning [15, 18] – one can add an all-1 vector (normalized by  $1/\sqrt{n}$ ) upon the  $n$  standard basis vectors. However, as far as we have concerned, the existing recovery algorithms in incoherent dictionary learning use either linear programming or OMP, which, again, are very time-inefficient on large datasets.

**Roadmap.** The rest of the paper is organized as follows. After giving some preliminaries (Section 2), we present our bias-aware sketches, analyze their performance, and illustrate their efficient implementations in the streaming model for real-time queries (Section 3). We then empirically verify the effectiveness of our algorithms via the five basic queries listed above (Section 4).

Table 1: List of notations

$\kappa$	number of sites
$[n]$	$[n] = \{1, 2, \dots, n\}$
$(\mathbf{x})_i$ or $x_i$	for $\mathbf{x} \in \mathbb{R}^n$ , both $(\mathbf{x})_i$ and $x_i$ represent the $i$ -th coordinate of $\mathbf{x}$
$\ \mathbf{x}\ _p$	$\ \mathbf{x}\ _p = (\sum_i  x_i ^p)^{\frac{1}{p}}$ for $\mathbf{x} = (x_1, \dots, x_n)$ ; when $p = \infty$ , $\ \mathbf{x}\ _\infty = \max_i  x_i $
$k$ -sparse	$\mathbf{x} \in \mathbb{R}^n$ is $k$ -sparse if $\mathbf{x}$ has <i>at most</i> $k$ non-zero coordinates
$\mathcal{S}_m(\mathbf{x})$	set of vectors in $\mathbb{R}^m$ obtained by choosing $m$ ( $\leq n$ ) coordinates from $\mathbf{x} \in \mathbb{R}^n$
$\text{Err}_p^k(\mathbf{x})$	$\text{Err}_p^k(\mathbf{x}) = \min_{k\text{-sparse } \mathbf{x}'} \ \mathbf{x} - \mathbf{x}'\ _p$
$\mathbf{x} - \beta$	for $\mathbf{x} \in \mathbb{R}^n, \beta \in \mathbb{R}$ , $\mathbf{x} - \beta = (x_1 - \beta, \dots, x_n - \beta)$
$\text{mean}(\mathbf{x})$	for $\mathbf{x} \in \mathbb{R}^n$ , $\text{mean}(\mathbf{x}) = \frac{1}{n} \sum_i x_i$
$\text{median}(\mathbf{x})$	for $\mathbf{x} \in \mathbb{R}^n$ , $\text{median}(\mathbf{x}) = x_{\frac{n+1}{2}}$ for odd $n$ , $\text{median}(\mathbf{x}) = (x_{\frac{n}{2}} + x_{\frac{n}{2}+1})/2$ for even $n$
$\text{argmin}_\beta f(\beta)$	$\text{argmin}_\beta f(\beta) = \{\alpha \mid f(\alpha) = \min_\beta f(\beta)\}$
$\sigma^2(\mathbf{x})$	variance of $\mathbf{x} \in \mathbb{R}^n$ ; $\sigma^2(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n (x_i - \text{mean}(\mathbf{x}))^2$
$\sigma^2(Y)$	variance of a random variable $Y$ ; $\sigma^2(Y) = \mathbf{E}[(Y - \mathbf{E}[Y])^2]$
$\Pi$	CM-Matrix. See Definition 1
$\Psi$	CS-Matrix. See Definition 2
$\Upsilon$	Sampling matrix. See Definition 3

## 2 Preliminaries

**Notations.** We summarize the main notations in this paper in Table 1. A quick scan of the table may be useful since some of the notations are not standard (e.g., a vector minus a scalar value:  $\mathbf{x} - \beta$ ).

**Count-Median.** The Count-Median algorithm [11] was originally designed for (approximately) finding frequent items in a data stream. In this paper we will use it as a component in our bias-aware recovery algorithms. We first introduce the *Count-Median* matrix.

**Definition 1 (CM-matrix)** Let  $h : [n] \rightarrow [s]$  be a hash function. A CM-matrix  $\Pi(h) \in \{0, 1\}^{s \times n}$  is defined as

$$\Pi(h)_{i,j} = \begin{cases} 1 & h(j) = i \\ 0 & h(j) \neq i. \end{cases}$$

For a vector  $\mathbf{x} \in \mathbb{R}^n$ , the following theorem shows that we can recover each coordinate of  $\mathbf{x}$  with a bounded error from  $\Theta(\log n)$  random sketching vectors  $\Pi(h)\mathbf{x}$ .

**Theorem 1 ([11])** Set  $s = \Theta(k/\alpha)$  for an  $\alpha \in (0, 1)$  and  $d = \Theta(\log n)$ . Let  $h^1, \dots, h^d : [n] \rightarrow [s]$  be  $d$  independent random hash functions, and let  $\Pi(h^1), \dots, \Pi(h^d)$  be the corresponding CM-matrices. Let  $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_n)$  be a vector such that

$$\hat{x}_j = \text{median}_{i \in [d]} \left\{ (\Pi(h^i) \mathbf{x})_{h^i(j)} \right\}.$$

We have

$$\Pr \left[ \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty \leq \alpha/k \cdot \text{Err}_1^k(\mathbf{x}) \right] \geq 1 - 1/n.$$

**Count-Sketch.** The Count-Sketch algorithm [6] is another well-known algorithm for finding frequent items in a data stream. Count-Sketch is similar to Count-Median; the main difference is that it introduces random signs in the sketching matrix.

**Definition 2 (CS-Matrix)** Let  $h : [n] \rightarrow [s]$  be a hash function, and  $r : [n] \rightarrow \{-1, 1\}$  be a random sign function. A CS-matrix  $\Psi(h, r) \in \{0, 1\}^{s \times n}$  is defined as

$$\Psi(h, r)_{i,j} = \begin{cases} r(j) & h(j) = i \\ 0 & h(j) \neq i. \end{cases}$$

Similarly, for a vector  $\mathbf{x} \in \mathbb{R}^n$ , we can recover each coordinate of  $\mathbf{x}$  with a bounded error from  $\Theta(\log n)$  sketching vectors  $\Psi(h, r)\mathbf{x}$ .

**Theorem 2 ([6])** Set  $s = \Theta(k/\alpha)$  for an  $\alpha \in (0, 1)$  and  $d = \Theta(\log n)$ . Let  $h^1, \dots, h^d : [n] \rightarrow [s]$  be  $d$  independent random hash functions, let  $r^1, \dots, r^d : [n] \rightarrow \{-1, 1\}$  be  $d$  independent random sign functions, and let  $\Psi(h^1, r^1), \dots, \Psi(h^d, r^d)$  be the corresponding CS-matrices. Let  $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_n)$  be a vector such that

$$\hat{x}_j = \text{median}_{i \in [d]} \left\{ r^i(j) \cdot (\Psi(h^i, r^i) \mathbf{x})_{h^i(j)} \right\}.$$

We have

$$\Pr \left[ \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty \leq \alpha/\sqrt{k} \cdot \text{Err}_2^k(\mathbf{x}) \right] \geq 1 - 1/n.$$

**Sampling Matrix.** We will use the following sampling matrix.

**Definition 3 (Sampling Matrix)** Let  $\Upsilon \in \{0, 1\}^{t \times n}$  be a 0/1 matrix by independently setting for each of the  $t$  rows exact one random coordinate to be 1.

**Chernoff Bound.** Let  $X_1, \dots, X_n$  be independent Bernoulli random variables such that  $\Pr[X_i = 1] = p_i$ . Let  $X = \sum_{i \in [n]} X_i$ . Let  $\mu = \mathbf{E}[X]$ . It holds that  $\Pr[X \geq (1 + \delta)\mu] \leq e^{-\delta^2 \mu/3}$  and  $\Pr[X \leq (1 - \delta)\mu] \leq e^{-\delta^2 \mu/2}$  for any  $\delta \in (0, 1)$ .

### 3 Bias-Aware Sketches

In this section we propose two efficient bias-aware sketches achieving  $\ell_\infty/\ell_2$ -guarantee and  $\ell_\infty/\ell_1$ -guarantee respectively.

### 3.1 Warm Up

The core of our algorithms is to estimate the bias of the input data. Before presenting our algorithms, we first discuss a few natural approaches that do *not* quite work, and then illustrate the high level ideas of our algorithms.

**Using Mean As the Bias.** The first natural idea is to use the mean of the input vector  $\mathbf{x}$ . However, this cannot lead to any theoretical error guarantee. Consider the following vector:  $\mathbf{x} = (\infty, \infty, 50, 50, 50, 50, 50, 50, 50, 50, 50)$  where  $\infty$  denotes a very large number, and  $k$  is set to be 2. The mean of the coordinates of  $\mathbf{x}$  is  $\infty$ , but the best bias value is  $\beta = 50$  which leads to a tail error 0 (RHS of (4)). Nevertheless, using the mean as the bias may work well in datasets where there are not many extreme values. We will show in our experiments (Section 4) that this is indeed the case for many real-world datasets.

**Searching the Bias in A Post-processing.** Another natural idea is to search the best bias value  $\beta$  in a post-processing procedure after performing the existing sketching algorithms such as Count-Sketch and Count-Median. This idea looks attractive since we can just reuse the existing sketching algorithms. For example, we may use *binary* search if the error term for  $p = 1$  or 2 is a convex function of  $\beta$  (which is not necessarily true). Another problem is, it does not fit the streaming setting where we want to answer queries in real-time. Indeed, every time when a query comes (e.g., a point query asking the value of  $x_i$ ), we have to binary-search  $\beta$  by computing the RHS of (4) at least a logarithmic number of times and then picking the best  $\beta$  value that minimize the error, which is very time consuming.

**Ideas of Our Algorithms.** In this paper we propose two simple yet efficient algorithms to achieve the error guarantee in (4), for  $p = 1$  and  $p = 2$  respectively. Our algorithms do not need a post-processing and can thus answer real-time queries in the streaming model. For  $p = 1$ , we compute by sampling an approximate median (denoted by  $med$ ) of coordinates in  $\mathbf{x}$ , and use it as the bias. Using the stability of median we can show that  $med$  is also an approximate median of the vector  $\mathbf{x}^*$  obtained from  $\mathbf{x}$  by dropping the  $k$  “outliers”. For  $p = 2$ , the idea is still to use the mean. However, as we have discussed previously, directly using the mean of all items will not give the desired theoretical guarantee, since the mean can be “contaminated” by the outliers (extreme values). We thus choose to employ a CM sketch and use the mean of the “middle” buckets in the CM sketch as the bias. Both algorithms are conceptually very simple, but the complete analysis turns out to be quite non-trivial. The next two subsections detail our algorithms.

### 3.2 Recovery with $\ell_\infty/\ell_1$ -Guarantee

In this section we give a sketching and recovery scheme with  $\ell_\infty/\ell_1$ -guarantee. That is, we try to design a sketching matrix  $\Phi \in \mathbb{R}^{t \times n}$  ( $t \ll n$ ) such that from  $\Phi\mathbf{x}$  we can recover an  $\hat{\mathbf{x}}$  satisfying  $\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O(1/k) \cdot \min_\beta \text{Err}_1^k(\mathbf{x} - \beta)$ .

#### 3.2.1 Algorithms

The sketching and recovery algorithms for  $\ell_\infty/\ell_1$  guarantees are described in Algorithm 1 and Algorithm 2 respectively. We again assume that the two algorithms can jointly sample for free hash functions  $h^1, \dots, h^d$ , and show how to do these in the distributed and streaming models later.

Similar to the  $\ell_\infty/\ell_2$  case, the first task is to estimate the best  $\beta$  that minimizes  $\text{Err}_1^k(\mathbf{x} - \beta)$ . This time we simply use sampling, that is, we sample  $\Theta(\log n)$  coordinates from  $\mathbf{x}$  and take the median, which we will show is good for the  $\ell_\infty/\ell_1$  guarantee. The final (implicit) sketching matrix  $\Phi$  is a vertical concatenation of  $d = \Theta(\log n)$  independent CM-matrix  $\Pi(h^i)$ ’s and the sampling matrix  $\Upsilon$ .

---

**Algorithm 1:**  $\ell_1$ -SKETCH( $\mathbf{x}$ )

---

**Input:**  $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$   
**Output:** sketch of  $\mathbf{x}$  and a set  $S \subseteq \{x_1, \dots, x_n\}$   
/\* assume  $s = c_s k$  for a constant  $c_s \geq 4$ ;  $d = \Theta(\log n)$ ;  $h^1, \dots, h^d : [n] \rightarrow [s]$  are common knowledge \*/  
1 generate a sampling matrix  $\Upsilon \in \{0, 1\}^{20 \log n \times n}$   
2  $\forall i \in [d], \mathbf{y}^i \leftarrow \Pi(h^i) \mathbf{x}$   
3  $S \leftarrow \Upsilon \mathbf{x}$   
4 **return**  $S, \{\mathbf{y}^1, \dots, \mathbf{y}^d\}$

---

---

**Algorithm 2:**  $\ell_1$ -RECOVER( $S, \{\mathbf{y}^1, \dots, \mathbf{y}^d\}$ )

---

**Input:**  $S$ : a set of randomly sampled coordinates of  $\mathbf{x}$ ;  $\{\mathbf{y}^i = \Pi(h^i) \mathbf{x} \mid i \in [d]\}$   
**Output:**  $\hat{\mathbf{x}}$  as an approximation of  $\mathbf{x}$   
/\* assume  $s = c_s k$  for a constant  $c_s \geq 4$ ;  $d = \Theta(\log n)$ ;  $h^1, \dots, h^d : [n] \rightarrow [s]$  are common knowledge \*/  
1  $\hat{\beta} \leftarrow$  median of coordinates in  $S$   
2  $\forall i \in [d], \boldsymbol{\pi}^i \leftarrow$  coordinate-wise sum of columns of  $\Pi(h^i)$   
3  $\forall i \in [d], \tilde{\mathbf{y}}^i \leftarrow \mathbf{y}^i - \hat{\beta} \boldsymbol{\pi}^i$   
/\* Run Count-Median recovery \*/  
4  $\forall j \in [n], \hat{z}_j \leftarrow \text{median}_{i \in [d]} \left\{ (\tilde{\mathbf{y}}^i)_{h^i(j)} \right\}$   
5  $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{z}} + \hat{\beta}$   
6 **return**  $\hat{\mathbf{x}}$

---

In the recovery phase we use Count-Median to recover  $\hat{\mathbf{z}}$  as an approximation to the *de-biased* vector  $\mathbf{x} - \hat{\beta}$ ; consequently  $\hat{\mathbf{z}} + \hat{\beta}$  will be a good approximation to  $\mathbf{x}$ .

The following theorem summarizes the performance of Algorithm 1 and Algorithm 2.

**Theorem 3** *There exists a sketching-recovery scheme such that for any  $\mathbf{x} \in \mathbb{R}^n$ , it computes  $\Phi \mathbf{x}$ , and then recovers an  $\hat{\mathbf{x}}$  as an approximation to  $\mathbf{x}$  from  $\Phi \mathbf{x}$  satisfying the following.*

$$\Pr[\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty \leq C_1/k \cdot \min_{\beta} \text{Err}_1^k(\mathbf{x} - \beta)] \geq 1 - C_2/n,$$

where  $C_1, C_2 > 0$  are two universal constants. The sketch can be constructed in time  $O(n \log n)$ ; the sketch size is bounded by  $O(k \log n)$ ; the recovery can be done in time  $O(n \log n)$ .

As mentioned in the introduction, we can convert  $\ell_\infty/\ell_1$  guarantee to  $\ell_1/\ell_1$  guarantee.

**Corollary 1** *The  $\hat{\mathbf{x}}$  recovered in Theorem 3 also guarantees*

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_1 = O(1) \cdot \min_{\beta} \text{Err}_1^k(\mathbf{x} - \beta)$$

with probability  $1 - O(1/n)$ .



### 3.2.2 Analysis

**Correctness.** Our first step is to get a better understanding of the best  $\beta$  that minimizes  $\text{Err}_1^k(\mathbf{x} - \beta)$ .

**Lemma 1** *Given  $\mathbf{x} \in \mathbb{R}^n$ , pick any  $\bar{\beta} \in \text{argmin}_{\beta} \text{Err}_1^k(\mathbf{x} - \beta)$ . Let  $\mathbf{x}^* \in \mathcal{S}_{n-k}(\mathbf{x})$  be the vector obtained by dropping the  $k$  coordinates that deviate the most from  $\bar{\beta}$ , we must have*

$$\|\mathbf{x}^* - \bar{\beta}\|_1 = \|\mathbf{x}^* - \text{median}(\mathbf{x}^*)\|_1.$$

**Proof:** For convenience we assume that  $(n - k)$  is odd, and then  $\|\mathbf{x}^* - \beta\|_1$  reaches the minimum only when  $\beta = \text{median}(\mathbf{x}^*)$ . It is easy to verify that our lemma also holds when  $(n - k)$  is even. Under this assumption, we only need to show  $\bar{\beta} = \text{median}(\mathbf{x}^*)$ .

We prove by contradiction. Suppose  $\bar{\beta} \neq \text{median}(\mathbf{x}^*)$ , then

$$\text{Err}_1^k(\mathbf{x} - \text{median}(\mathbf{x}^*)) \leq \|\mathbf{x}^* - \text{median}(\mathbf{x}^*)\|_1 < \text{Err}_1^k(\mathbf{x} - \bar{\beta}),$$

contradicting the definition of  $\bar{\beta}$ . ■

Our next lemma shows that the median of the sampled coordinates  $S$  will not deviate from the median of  $\mathbf{x}$  by much.

**Lemma 2** *Given a vector  $\mathbf{x} \in \mathbb{R}^n$  with its coordinates sorted non-decreasingly:  $x_1 \leq x_2 \leq \dots \leq x_n$ , if we randomly sample with replacement  $t = 20 \log n$  coordinates from  $\mathbf{x}$ , then with probability at least  $1 - 1/n$  the median of the  $t$  samples falls into the range  $[x_{n/2-n/6}, x_{n/2+n/6}]$ .*

**Proof:** Let  $X_1, \dots, X_t$  be the samples we pick. The median of them does not fall into the range  $[x_{n/2-n/6}, x_{n/2+n/6}]$  if and only if one of the following events happens,

- $\mathcal{E}_1$ : at least half of the samples larger than  $x_{n/2+n/6}$ .
- $\mathcal{E}_2$ : at least half of the samples smaller than  $x_{n/2-n/6}$ .

We first bound the probability that  $\mathcal{E}_1$  happens. Let  $Y_i$  be the random variables such that  $Y_i = 1$  if  $X_i > x_{n/2+n/6}$ , and  $Y_i = 0$  otherwise. We have  $\mathbb{E} [\sum_{i=1}^t Y_i] = t/3$ . By a Chernoff bound, we have

$$\begin{aligned} & \Pr \left[ \sum_{i=1}^t Y_i - \frac{t}{3} > \frac{t}{6} \right] \\ & \leq \exp \left( -\frac{t}{12} \right) \\ & < 1/(2n). \quad (t = 20 \log n) \end{aligned}$$

Similarly we can show that the probability that  $\mathcal{E}_2$  happens is at most  $1/(2n)$ . The lemma follows. ■

The following lemma shows any coordinate that is close to the median of coordinates in  $\mathbf{x} = (x_1, \dots, x_m)$  can be used to approximate the best  $\beta$  that minimizes  $\sum_{i \in [m]} |x_i - \beta|$ .

**Lemma 3** *Given a vector  $\mathbf{x} \in \mathbb{R}^m$  with its coordinates sorted non-decreasingly:  $x_1 \leq x_2 \leq \dots \leq x_m$ , for any  $j$  such that  $\frac{m}{4} < j < \frac{3m}{4}$ , we have*

$$\sum_{i \in [m]} |x_i - x_j| \leq 2 \cdot \min_{\beta} \sum_{i \in [m]} |x_i - \beta|.$$

**Proof:** For simplicity we assume  $m$  is odd; the even case can be handled similarly. Let  $t = (m + 1)/2$  be the index of the median coordinate. If  $j = t$  then we are done. Otherwise, w.l.o.g., we assume  $j < t$ . We have

$$\begin{aligned}
& \left( \sum_{i=1}^m |x_i - x_j| \right) - \left( \sum_{i=1}^m |x_i - x_t| \right) \\
&= \sum_{i=1}^{t-j} i \cdot (x_{t-i+1} - x_{t-i}) \\
&= -(t-j) \cdot x_j + \sum_{i=j+1}^t x_i \\
&= \sum_{i=j+1}^t (x_i - x_j) \\
&\leq \sum_{i=1}^{m/4} (x_t - x_i) \quad \left( \text{since } \frac{m}{4} < j < t = \frac{m+1}{2} \right) \\
&\leq \sum_{i=1}^m |x_i - x_t|.
\end{aligned}$$

The lemma follows. ■

Now we are ready to prove the correctness of Theorem 3, that is,

$$\Pr \left[ \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O\left(\frac{1}{k}\right) \cdot \min_{\beta} \text{Err}_1^k(\mathbf{x} - \beta) \right] = 1 - O\left(\frac{1}{n}\right). \quad (6)$$

W.l.o.g. we assume the coordinates of  $\mathbf{x}$  are sorted as  $x_1 \leq x_2 \leq \dots \leq x_n$ . To simplify the discussion, we assume  $t$  at Line 3 of Algorithm 1 is odd. The even case can be verified similarly.

Let  $\hat{\beta}$  be the median of the  $t$  samples in  $S$  (Line 1 in Algorithm 2). Let  $\alpha \in \arg\min_{\beta} \text{Err}_1^k(\mathbf{x} - \beta)$ . Let  $\mathbf{x}^*$  be the vector obtained by dropping the  $k$  coordinates from  $\mathbf{x}$  that deviate the most from  $\alpha$ .

By Lemma 2,  $\hat{\beta} \in [x_{n/2-n/6}, x_{n/2+n/6}]$  holds with probability  $1 - 1/n$ . Note that we can assume that  $k = O(n/\log n)$  (otherwise the sketch can just be  $\mathbf{x}$  itself which has size  $O(k \log n)$ ). We thus have

$$\Pr \left[ (\mathbf{x}^*)_{\frac{(n-k)}{4}} \leq \hat{\beta} \leq (\mathbf{x}^*)_{\frac{3(n-k)}{4}} \right] > 1 - \frac{1}{n}. \quad (7)$$

Applying Lemma 3 to  $\mathbf{x}^*$  (with  $m = n - k$ ), with probability at least  $(1 - 1/n)$  it holds that

$$\begin{aligned}
\text{Err}_1^k(\mathbf{x} - \hat{\beta}) &\leq \|\mathbf{x}^* - \hat{\beta}\|_1 \\
&\leq 2 \cdot \min_{\beta} \|\mathbf{x}^* - \beta\|_1 \quad (\text{by (7) and Lemma 3}) \\
&= 2 \cdot \|\mathbf{x}^* - \text{median}(\mathbf{x}^*)\|_1 \\
&= 2 \cdot \|\mathbf{x}^* - \alpha\|_1 \quad (\text{by Lemma 1}) \\
&= 2 \cdot \min_{\beta} \text{Err}_1^k(\mathbf{x} - \beta),
\end{aligned} \quad (8)$$

---

**Algorithm 3:**  $\ell_2$ -SKETCH( $\mathbf{x}$ )

---

**Input:**  $\mathbf{x} \in \mathbb{R}^n$ **Output:** the sketch of  $\mathbf{x}$ 

*/\* assume  $s = c_s k$  for a constant  $c_s \geq 4$ ;  $d = \Theta(\log n)$ ;  $g, h^1, \dots, h^d : [n] \rightarrow [s]$ ;  
 $r^1, \dots, r^d : [n] \rightarrow \{-1, 1\}$  are common knowledge \*/*

```
1  $\mathbf{w} \leftarrow \Pi(g)\mathbf{x}$ 
2  $\forall i \in [d], \mathbf{y}^i \leftarrow \Psi(h^i, r^i)\mathbf{x}$ 
3 return  $\mathbf{w}, \{\mathbf{y}^1, \dots, \mathbf{y}^d\}$ 
```

---

where the last equality holds due to the definitions of  $\alpha$  and  $\mathbf{x}^*$ . By Theorem 1 (property of Count-Median) and Line 4 of Algorithm 2 we have

$$\Pr \left[ \|\hat{\mathbf{z}} - (\mathbf{x} - \hat{\beta})\|_\infty = O\left(\frac{1}{k}\right) \cdot \text{Err}_1^k(\mathbf{x} - \hat{\beta}) \right] \geq 1 - \frac{1}{n}.$$

Since at Line 5 we set  $\hat{\mathbf{x}} = \hat{\mathbf{z}} + \hat{\beta}$ , we have

$$\Pr \left[ \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O\left(\frac{1}{k}\right) \cdot \text{Err}_1^k(\mathbf{x} - \hat{\beta}) \right] \geq 1 - \frac{1}{n}. \quad (9)$$

Inequality (6) follows from (8) and (9).

**Complexities.** Since CM-matrix only has one non-zero entry in each column, using sparse matrix representation we can compute  $\Pi(h^i)\mathbf{x}$  ( $i \in [d]$ ) in  $O(n)$  time. Thus the sketching phase can be done in time  $O(nd) = O(n \log n)$ .

The sketch size is  $O(k \log n)$  since each  $\Psi(h^i)\mathbf{x}$  ( $i \in [d]$ ) has size  $O(k)$ .

In the recovery phase, the dominating cost is the computation of coordinates in  $\hat{\mathbf{z}}$ , for each of which we need  $O(d) = O(\log n)$  time. Thus the total cost is  $O(n \log n)$ .

### 3.3 Recovery with $\ell_\infty/\ell_2$ -Guarantee

We first give a scheme with an  $\ell_\infty/\ell_2$ -guarantee. That is, we try to design a sketching matrix  $\Phi \in \mathbb{R}^{t \times n}$  ( $t \ll n$ ) such that from  $\Phi\mathbf{x}$  we can recover an  $\hat{\mathbf{x}}$  satisfying  $\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O(1/\sqrt{k}) \cdot \min_\beta \text{Err}_2^k(\mathbf{x} - \beta)$ .

#### 3.3.1 Algorithms

Our sketching algorithm is described in Algorithm 3, and the recovery algorithm is described in Algorithm 4, which we will parse shortly.

For now we assume that the sketching algorithm and the recovery algorithm can jointly sample *for free* (1) independent random hash functions  $g, h^1, \dots, h^d : [n] \rightarrow [s]$ , and (2) independent random signed functions  $r^1, \dots, r^d : [n] \rightarrow \{-1, 1\}$ . We will specify how to implement these in the distributed and streaming models later.

The high level idea of our algorithms is that we first use the Count-Median matrix to obtain a good approximation  $\hat{\beta}$  of the  $\beta$  that minimizes  $\text{Err}_2^k(\mathbf{x} - \beta)$ , and then use the Count-Sketch algorithm to recover  $\hat{\mathbf{z}}$  as an approximation to the *de-biased* vector  $\mathbf{x} - \hat{\beta}$ ; and consequently  $\hat{\mathbf{z}} + \hat{\beta}$  will be a good approximation to  $\mathbf{x}$ . The final (implicit) sketching matrix  $\Phi \in \mathbb{R}^{s(d+1) \times n}$  in Algorithm 3 is a vertical concatenation of a CM-matrix  $\Pi(g)$  and  $d = \Theta(\log n)$  independent CS-matrices  $\Psi(h^i, r^i)$ 's.

---

**Algorithm 4:**  $\ell_2$ -RECOVER( $\mathbf{w}, \{\mathbf{y}^1, \dots, \mathbf{y}^d\}$ )

---

**Input:**  $\mathbf{w} = \Pi(g)\mathbf{x}$ ;  $\{\mathbf{y}^i = \Psi(h^i, r^i)\mathbf{x} \mid i \in [d]\}$   
**Output:**  $\hat{\mathbf{x}}$  as an approximation of  $\mathbf{x}$   
/\* assume  $s = c_s k$  for a constant  $c_s \geq 4$ ;  $d = \Theta(\log n)$ ;  $g, h^1, \dots, h^d : [n] \rightarrow [s]$ ;  
 $r^1, \dots, r^d : [n] \rightarrow \{-1, 1\}$  are common knowledge \*/  
1  $\boldsymbol{\pi} \leftarrow$  coordinate-wise sum of columns of  $\Pi(g)$   
2 w.l.o.g. assume  $w_1/\pi_1 \leq \dots \leq w_s/\pi_s$ ; set  $\hat{\beta} = \sum_{i=s/2-k}^{s/2+k-1} w_i / \sum_{i=s/2-k}^{s/2+k-1} \pi_i$   
3  $\forall i \in [d], \boldsymbol{\psi}^i \leftarrow$  coordinate-wise sum of columns of  $\Psi(h^i, r^i)$   
4  $\forall i \in [d], \tilde{\mathbf{y}}^i \leftarrow \mathbf{y}^i - \hat{\beta}\boldsymbol{\psi}^i$   
/\* Run the Count-Sketch recovery \*/  
5  $\forall j \in [n], \hat{z}_j \leftarrow \text{median}_{i \in [d]} \left\{ r^i(j) \cdot (\tilde{\mathbf{y}}^i)_{h^i(j)} \right\}$   
6  $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{z}} + \hat{\beta}$   
7 **return**  $\hat{\mathbf{x}}$ 

---

In Algorithm 4, to approximate the best  $\beta$  we first sum up all the columns of  $\Pi(g)$ , giving a vector  $\boldsymbol{\pi} = (\pi_1, \dots, \pi_s)$  (Line 1). Let  $\mathbf{w} = \Pi(g)\mathbf{x} \in \mathbb{R}^s$ . W.l.o.g. assume that  $w_1/\pi_1 \leq \dots \leq w_s/\pi_s$ . We estimate  $\beta$  by

$$\hat{\beta} = \sum_{i=s/2-k}^{s/2+k-1} w_i / \sum_{i=s/2-k}^{s/2+k-1} \pi_i.$$

The intuition of this estimation is the following. First note that  $w_i/\pi_i$  ( $i \in [s]$ ) is the average of coordinates of  $\mathbf{x}$  that are hashed into the  $i$ -th coordinate(bucket) of sketching vector  $\Pi(g)\mathbf{x}$ . In the case that there is no “outlier” coordinate of  $\mathbf{x}$  that is hashed into the  $i$ -th bucket of  $\Pi(g)\mathbf{x}$ ,  $w_i/\pi_i$  ( $i \in [s]$ ) should be close to the best bias  $\beta$ . Since there are at most  $k$  outliers, if we choose  $s \geq 4k$  then most of these  $s$  buckets in  $\Pi(g)$  will not be “contaminated” by outliers. The next idea is to sort the buckets according to the average of coordinates of  $\mathbf{x}$  hashed into it (i.e.,  $w_i/\pi_i$ ), and then choose the  $2k$  buckets around the median and take the average of coordinates hashed into those buckets (Line 2). We can show that the average of coordinates of  $\mathbf{x}$  that are hashed into these  $2k$  “median” buckets is a good estimation of the best  $\beta$ . Note that there could still be outliers hashed into the median  $2k$  buckets, but we are able to prove that such outliers will not affect the estimation of  $\beta$  by much. After getting an estimate of  $\beta$  we de-bias the sketching vector  $\mathbf{y}$  (Line 3 and 4) for the next step recovery (Line 5 and 6).

The following theorem summarizes the performance of Algorithm 3 and Algorithm 4. The analysis will be given in Section 3.3.2.

**Theorem 4** *There exists a sketching-recovery scheme such that for any  $\mathbf{x} \in \mathbb{R}^n$ , it computes  $\Phi\mathbf{x}$ , and then recovers an  $\hat{\mathbf{x}}$  as an approximation to  $\mathbf{x}$  from  $\Phi\mathbf{x}$  satisfying the following:*

$$\Pr[\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty \leq C_1/\sqrt{k} \cdot \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)] \geq 1 - C_2/n,$$

where  $C_1, C_2 > 0$  are two universal constants. The sketch can be constructed in time  $O(n \log n)$ ; the sketch size is bounded by  $O(k \log n)$ ; the recovery can be done in time  $O(n \log n)$ .

As mentioned in the introduction, we can convert  $\ell_\infty/\ell_2$  guarantee to  $\ell_2/\ell_2$  guarantee.

**Corollary 2** *The  $\hat{\mathbf{x}}$  recovered in Theorem 4 also guarantees*

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_2 = O(1) \cdot \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)$$

with probability  $1 - O(1/n)$ .

### 3.3.2 Analysis

**Correctness.** We first try to replace the somewhat obscure expression  $\min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)$  in Theorem 4 with another expression that is more convenient to use. Let  $\mathcal{S}_m(\mathbf{x})$  be set of vectors in  $\mathbb{R}^m$  obtained by choosing  $m$  ( $\leq n$ ) coordinates from  $\mathbf{x} \in \mathbb{R}^n$ .

**Lemma 4** *For any  $\mathbf{x} \in \mathbb{R}^n$  and  $k < n$ , let  $\mathbf{x}^*$  be a vector in  $\mathcal{S}_{n-k}(\mathbf{x})$  that has the minimum variance. It holds that*

$$\begin{aligned} \left( \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta) \right)^2 &= (n - k) \sigma^2(\mathbf{x}^*) \\ &= \|\mathbf{x}^* - \text{mean}(\mathbf{x}^*)\|_2^2. \end{aligned} \tag{10}$$

Furthermore,  $\mathbf{x}^*$  is equivalent to the vector obtained by dropping the  $k$  coordinates from  $\mathbf{x}$  that deviate the most from  $\text{mean}(\mathbf{x}^*)$ .

**Proof:** First, by the definition of  $\mathbf{x}^*$  we have

$$(n - k) \sigma^2(\mathbf{x}^*) = \min_{\mathbf{x}' \in \mathcal{S}_{n-k}(\mathbf{x})} \|\mathbf{x}' - \text{mean}(\mathbf{x}')\|_2^2.$$

By the definition of  $\text{Err}_2^k(\cdot)$ , we have

$$\begin{aligned} \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta) &\geq \min_{\mathbf{x}' \in \mathcal{S}_{n-k}(\mathbf{x})} \min_{\beta} \|\mathbf{x}' - \beta\| \\ &= \min_{\mathbf{x}' \in \mathcal{S}_{n-k}(\mathbf{x})} \|\mathbf{x}' - \text{mean}(\mathbf{x}')\|_2. \end{aligned}$$

Thus to prove (10), it suffices to show that

$$\min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta) \leq \min_{\mathbf{x}' \in \mathcal{S}_{n-k}(\mathbf{x})} \|\mathbf{x}' - \text{mean}(\mathbf{x}')\|_2.$$

Since the order of the coordinates in  $\mathbf{x}$  do not matter, w.l.o.g. we assume  $\mathbf{x}^* = (x_1, x_2, \dots, x_{n-k})$ . Let  $\gamma = \text{mean}(\mathbf{x}^*)$ , and write  $x_i = \gamma + \Delta_i$ , or equivalently,  $\Delta_i = x_i - \gamma$ . Note that if

$$\min_{i \in [n] \setminus [n-k]} |\Delta_i| \geq \max_{i \in [n-k]} |\Delta_i|, \tag{11}$$

then we are done because

$$\begin{aligned} \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)^2 &\leq \text{Err}_2^k(\mathbf{x} - \gamma)^2 \\ &= \sum_{i \in [n-k]} \Delta_i^2 = \|\mathbf{x}^* - \gamma\|_2^2. \end{aligned} \tag{12}$$

Now we assume (11) is false. Again w.l.o.g., we assume  $|\Delta_1| = \max_{i \in [n-k]} |\Delta_i|$  and  $|\Delta_n| = \min_{i \in [n] \setminus [n-k]} |\Delta_i|$ , then  $|\Delta_1| > |\Delta_n|$ . Let  $\mathbf{x}' = (x_2, x_3, \dots, x_{n-k-1}, x_{n-k}, x_n) \in \mathcal{S}_{n-k}$ , that is,  $\mathbf{x}'$  is obtained by dropping  $x_i$  from  $\mathbf{x}^*$  and then appending  $x_n$ , we have

$$\begin{aligned}
\|\mathbf{x}' - \text{mean}(\mathbf{x}')\|_2^2 &\leq \|\mathbf{x}' - \text{mean}(\mathbf{x}^*)\|_2^2 \\
&= \|\mathbf{x}' - \gamma\|_2^2 \quad (\text{by definition of } \gamma) \\
&= \Delta_n^2 + \sum_{i=2}^{n-k} \Delta_i^2 \quad (\text{by definition of } \Delta_i) \\
&< \Delta_1^2 + \sum_{i=2}^{n-k} \Delta_i^2 \\
&= \|\mathbf{x}^* - \text{mean}(\mathbf{x}^*)\|_2^2,
\end{aligned}$$

which contradicts the definition of  $\mathbf{x}^*$ . Hence (11) holds, and consequently (10) holds.

On the other hand, (11) also implies that  $x_{n-k+1}, \dots, x_n$  are the  $k$  coordinates of  $\mathbf{x}$  that deviate the most from  $\gamma = \text{mean}(\mathbf{x}^*)$ . ■

We next show that a good approximation of  $\gamma = \text{mean}(\mathbf{x}^*)$  is also a good approximation of the best  $\beta$ .

**Lemma 5** *For any  $\mathbf{x} \in \mathbb{R}^n$  and  $k < n$ , let  $\mathbf{x}^*$  be a vector in  $\mathcal{S}_{n-k}(\mathbf{x})$  that has the minimum variance. For any  $\alpha$  such that  $|\text{mean}(\mathbf{x}^*) - \alpha|^2 \leq C \cdot \sigma^2(\mathbf{x}^*)$  for any constant  $C > 0$ , we have*

$$\text{Err}_2^k(\mathbf{x} - \alpha)^2 = O\left(\min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)^2\right).$$

**Proof:** W.l.o.g. we again assume  $\mathbf{x}^* = (x_1, \dots, x_{n-k})$ . Define  $f(b) \triangleq \|\mathbf{x}^* - b\|_2^2$ . Let  $\gamma = \text{mean}(\mathbf{x}^*)$ . By Lemma 4 we have

$$f(\gamma) = (n-k)\sigma^2(\mathbf{x}^*) = \|\mathbf{x}^* - \gamma\|_2^2 = \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)^2. \quad (13)$$

Write  $\alpha = \gamma + \Delta$  and thus  $\Delta^2 \leq C\sigma^2(\mathbf{x}^*)$ ,

$$\begin{aligned}
\text{Err}_2^k(\mathbf{x} - \alpha)^2 &\leq \|\mathbf{x}^* - \alpha\|_2^2 \\
&= f(\alpha) = f(\gamma + \Delta) \\
&= \sum_{i \in [n-k]} ((x_i - \gamma) - \Delta)^2 \\
&= (n-k)\Delta^2 + \sum_{i=1}^{n-k} (x_i - \gamma)^2 - 2\Delta \sum_{i=1}^{n-k} (x_i - \gamma) \\
&\leq (n-k) \cdot C\sigma^2(\mathbf{x}^*) + \|\mathbf{x}^* - \gamma\|_2^2 + 0 \\
&= O\left(\min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)^2\right). \quad (\text{by (13)})
\end{aligned}$$

We are done. ■

The following lemma shows that the approximation  $\hat{\beta}$  obtained by the recovery algorithm (Algorithm 4) is close to  $\text{mean}(\mathbf{x}^*)$ .

**Lemma 6** Let  $\hat{\beta}$  be given at Line 2 of Algorithm 4. If  $s = c_s k$  for a sufficiently large constant  $c_s \geq 4$ , it holds that

$$\Pr \left[ \left( \hat{\beta} - \text{mean}(\mathbf{x}^*) \right)^2 = O(\sigma^2(\mathbf{x}^*)) \right] = 1 - O\left(\frac{1}{n}\right)$$

for any  $\mathbf{x}^*$  in  $\mathcal{S}_{n-k}(\mathbf{x})$  that has the minimum variance.

Before proving Lemma 6, we show a bound on the difference between the average of all coordinates of a vector and the average of a subset of coordinates.

**Lemma 7** Let  $\mathbf{x} = \{x_1, \dots, x_m\} \in \mathbb{R}^m$  be a vector. Let  $S$  be a subset of  $\mathbf{x}$ 's coordinates of size  $|S| = \Theta(m)$ . Let  $\mu = \frac{1}{m} \sum_{i \in [m]} x_i$ , and  $\mu' = \frac{1}{|S|} \sum_{i \in S} x_i$ . Then we have

$$|\mu' - \mu|^2 = O(\sigma^2(\mathbf{x})).$$

**Proof:** W.l.o.g., let  $x_1 \leq \dots \leq x_m$ . Let  $\alpha \in (0, 1)$  be an arbitrary constant. Let

$$\mu_1 = \frac{1}{\alpha m} \sum_{i \in [\alpha m]} x_i \quad \text{and} \quad \mu_2 = \frac{1}{(1-\alpha)m} \sum_{i \in [m] \setminus [\alpha m]} x_i.$$

We only need to prove two extreme cases where  $S = [\alpha m]$  or  $S = [m] \setminus [\alpha m]$  (in both cases  $|S| = \Theta(n)$  since  $\alpha = (0, 1)$  is an arbitrary constant):

$$|\mu_1 - \mu|^2 \leq \frac{1}{\alpha} \cdot \sigma^2(\mathbf{x}) \quad \text{and} \quad |\mu_2 - \mu|^2 \leq \frac{1}{1-\alpha} \cdot \sigma^2(\mathbf{x}). \quad (14)$$

For simplicity (and w.o.l.g.), we assume  $\mu = 0$ , since we can always define  $y_i = x_i - \mu$  and prove on  $y_i$ 's. We can write the variance of  $\mathbf{x}$  as

$$\begin{aligned} \sigma^2(\mathbf{x}) &= \frac{1}{m} \left( \sum_{i \in [\alpha m]} x_i^2 + \sum_{i \in [m] \setminus [\alpha m]} x_i^2 \right) \\ &\geq (\alpha m \mu_1^2 + (1-\alpha)m \mu_2^2) / m \quad (\text{Cauchy-Schwarz}) \\ &= \alpha \mu_1^2 + (1-\alpha) \mu_2^2. \end{aligned}$$

(14) follows straightforwardly. ■

**Proof:** (for Lemma 6) Fix any  $\mathbf{x}^* = (x_{i_1}, \dots, x_{i_{n-k}})$  in  $\mathcal{S}_{n-k}(\mathbf{x})$  that has the minimum variance. Let  $O$  be the set of the top- $k$  indices  $i$  in  $\mathbf{x}$  that maximize  $|x_i - \text{mean}(\mathbf{x}^*)|$ . By Lemma 4 we have that  $\mathbf{x}^*$  can be obtained from  $\mathbf{x}$  by dropping coordinates indexed by  $O$ .

We call an index  $i \in [s]$  in the sketching vector  $\mathbf{w} = \Pi(g)\mathbf{x}$  *contaminated* if there is at least one  $o \in O$  such that  $g(o) = i$ . W.l.o.g., we assume  $w_1/\pi_1 \leq \dots \leq w_s/\pi_s$  where  $\pi$  is defined at Line 1 of Algorithm 4. Let

$$I = \{i \mid s/2 - k \leq i < s/2 + k\}$$

be the  $2k$  “median” indices of  $\mathbf{w}$ , and

$$\bar{I} = \{i \mid i < s/2 - k \vee i \geq s/2 + k\}$$

be the rest of indices in  $\mathbf{w}$ . Since  $|O| = k$  and  $s \geq 4k$ , there are *at most*  $k$  coordinates in  $I$  that are contaminated, and *at least*  $k$  coordinates in  $\bar{I}$  that are *not* contaminated.

The approximation to the bias  $\beta$  at Line 2 of Algorithm 4 can be written as

$$\hat{\beta} = \frac{\sum_{i \in I} w_i}{\sum_{i \in I} \pi_i}. \quad (15)$$

Let  $O' = I \cap g(O)$  be the indices in  $I$  that are contaminated, and let  $J$  be an arbitrary subset of  $\bar{I}$  with size  $|O'|$ . Define

$$\gamma_1 = \min_J \frac{\sum_{i \in I \cup J \setminus O'} w_i}{\sum_{i \in I \cup J \setminus O'} \pi_i} \quad \text{and} \quad \gamma_2 = \max_J \frac{\sum_{i \in I \cup J \setminus O'} w_i}{\sum_{i \in I \cup J \setminus O'} \pi_i}. \quad (16)$$

It is easy to see that  $\gamma_1 \leq \hat{\beta} \leq \gamma_2$ : since  $s \geq 4k$ , one can always find a subset  $J \subseteq \bar{I}$  of size  $|O'|$  such that for any  $j \in J, o \in O'$  we have  $w_j/\pi_j \geq w_o/\pi_o$ , and replacing  $O'$  with  $J$  only increases the RHS of (15); On the other hand one can also find a subset  $J \subseteq \bar{I}$  of size  $|O'|$  such that for any  $j \in J, o \in O'$  we have  $w_j/\pi_j \leq w_o/\pi_o$ , and replacing  $O'$  with  $J$  only decreases the RHS of (15).

We now show that both  $\gamma_1$  and  $\gamma_2$  deviate at most  $O(\sigma(\mathbf{x}^*))$  from  $\text{mean}(\mathbf{x}^*)$ , and consequently  $\hat{\beta}$ , which is sandwiched by  $\gamma_1$  and  $\gamma_2$ , deviates from  $\text{mean}(\mathbf{x}^*)$  by at most  $O(\sigma(\mathbf{x}^*))$ . Consider the set  $G = g^{-1}(I \cup J \setminus O')$ . First, by definitions of  $I, J$  and  $O'$  we have  $G \subseteq \{i_1, \dots, i_{n-k}\}$ ; and thus  $\{x_j \mid j \in G\}$  are coordinates in  $\mathbf{x}^*$ . Second, since  $|I \cup J \setminus O'| = \Theta(k)$  and  $g$  is a random mapping from  $[n]$  to  $[s]$ , by a Chebyshev inequality we have  $|G| = \Theta(n)$  with probability at least  $1 - O(1/n)$ .<sup>2</sup> For any  $J \subseteq \bar{I}$  of size  $|O'|$ , let

$$\gamma_J = \frac{1}{|G|} \sum_{j \in G} x_j = \frac{\sum_{i \in I \cup J \setminus O'} w_i}{\sum_{i \in I \cup J \setminus O'} \pi_i}.$$

By Lemma 7, we have

$$|\gamma_J - \text{mean}(\mathbf{x}^*)| = O(\sigma(\mathbf{x}^*)). \quad (17)$$

Since Inequality (17) applies to any  $J \subseteq \bar{I}$  of size  $|O'|$ , we have  $|\gamma - \text{mean}(\mathbf{x}^*)| = O(\sigma(\mathbf{x}^*))$  for any  $\gamma \in \{\gamma_1, \gamma_2\}$ .  $\blacksquare$

Now we are ready to prove the correctness of Theorem 4, that is,

$$\Pr \left[ \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O\left(\frac{1}{\sqrt{k}}\right) \cdot \min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta) \right] = 1 - O\left(\frac{1}{n}\right). \quad (18)$$

Let  $\mathbf{x}^*$  be a vector in  $\mathcal{S}_{n-k}(\mathbf{x})$  that has the minimum variance. At Line 4-5 in Algorithm 4 the Count-Sketch recovery algorithm is used to compute  $\hat{\mathbf{z}}$  as an approximation to  $\mathbf{x} - \hat{\beta}$ . By Theorem 2 we have

$$\Pr \left[ \|\hat{\mathbf{z}} - (\mathbf{x} - \hat{\beta})\|_\infty = O\left(\frac{1}{\sqrt{k}}\right) \cdot \text{Err}_2^k(\mathbf{x} - \hat{\beta}) \right] \geq 1 - \frac{1}{n}.$$

Since at Line 6 we set  $\hat{\mathbf{x}} = \hat{\mathbf{z}} + \hat{\beta}$ , it holds that

$$\Pr \left[ \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty = O\left(\frac{1}{\sqrt{k}}\right) \cdot \text{Err}_2^k(\mathbf{x} - \hat{\beta}) \right] \geq 1 - \frac{1}{n}. \quad (19)$$

By Lemma 6,

$$\Pr \left[ |\hat{\beta} - \text{mean } \mathbf{x}^*| = O(\sigma(\mathbf{x}^*)) \right] = 1 - O\left(\frac{1}{n}\right).$$

---

<sup>2</sup>More precisely, define for each  $i \in [n]$  a random variable  $Y_i$ , which is 1 if  $g(i) \in I \cup J \setminus O'$  and 0 otherwise. Since  $|I \cup J \setminus O'| = \Theta(k)$  and  $s = \Theta(k)$ , we have  $\mathbf{E}[Y_i] = \Theta(1)$ , and  $\mathbf{Var}[Y_i] \leq \mathbf{E}[Y_i^2] = O(1)$ . Next note that  $|G| = \sum_{i \in [n]} Y_i$ . We thus can apply a Chebyshev inequality on  $Y_i$ 's and conclude that  $|G| = \Theta(n)$  with probability  $1 - O(1/n)$ .



Plugging it to Lemma 5 we have with probability at least  $(1 - O(1/n))$  that

$$\text{Err}_2^k(\mathbf{x} - \hat{\beta}) = O\left(\min_{\beta} \text{Err}_2^k(\mathbf{x} - \beta)\right). \quad (20)$$

Inequality (18) follows from (19) and (20).

**Complexities.** Since each CS-Matrix or CM-matrix only has one non-zero entry in each column, using sparse matrix representation we can compute  $\Psi(h^i, r^i)\mathbf{x}$  ( $i \in [d]$ ) or  $\Pi(g)\mathbf{x}$  in  $O(n)$  time. Thus the sketching phase can be done in time  $O(nd) = O(n \log n)$ .

The sketch size is  $O(k \log n)$ , simply because  $\Pi(g)\mathbf{x}$  and each  $\Psi(h^i, r^i)\mathbf{x}$  ( $i \in [d]$ ) has size  $O(k)$ .

In the recovery phase, the dominating cost is the computation of coordinates in  $\hat{\mathbf{z}}$ , for each of which we need  $O(d) = O(\log n)$  time. Thus the total cost is  $O(n \log n)$ .

### 3.4 Streaming Implementations

Recall that in the streaming model, we need to process by a one-pass scan a real-time sequence of  $m$  (unknown at the beginning) items  $\{(e_1, \Delta_1), \dots, (e_m, \Delta_m)\}$  where  $e_i \in [n]$  is the ID of the  $i$ -th item and  $\Delta_i \in \mathbb{R}$  is the value change of  $e_i$ , using a small memory *space* which is much smaller than the size of the input (thus we cannot afford to maintaining the whole vector  $\mathbf{x}$ ). In the recovery problem, at the end of the streaming process we need to compute the implicit input vector  $\mathbf{x} \in \mathbb{R}^n$  where  $x_i = \sum_{j \in [m]: e_j = i} \Delta_j$ , using only the information kept in the memory. We refer readers to surveys [7, 23] for more background on streaming algorithms.

We now illustrate how to solve the recovery problem in the streaming model using linear sketches. Again by the linearity, to process each input item  $(e_i, \Delta_i)$  we can simply update our sketching vector as  $\Phi\mathbf{x} \leftarrow \Phi\mathbf{x} + \Phi\mathbf{z}$ , where  $\mathbf{z} \in \mathbb{R}^n$  is an all-0 vector except the  $e_i$ -th coordinate being  $\Delta_i$ . The space usage of this approach will be equal to the size of the sketch  $\Phi\mathbf{x}$  which is much smaller than  $n$ , and the update  $\Phi\mathbf{z}$  can be computed in time equal to the column-sparsity of  $\Phi$ , which is  $O(\log n)$  in our choices of  $\Phi$ .

In the rest of this section we discuss how to maintain the bias  $\beta$  at any time step in the streaming process. This is useful since we would like to answer point queries (i.e., given  $j \in [n]$ , return  $x_j$ ) in  $O(\log n)$  time without decoding the whole vector  $\mathbf{x}$ , and to do this we have to first recover  $\beta$  efficiently.

For the  $\ell_\infty/\ell_1$  guarantee we can easily maintain a good approximation of  $\beta$  with  $O(\log \log n)$  time per update: we can simply keep the  $\Theta(\log n)$  sampled coordinates sorted (e.g., using a balanced binary search tree) during the streaming process, and use their median as an approximation of  $\beta$  at any time step. For the  $\ell_\infty/\ell_2$  guarantee, the recovery procedure in Algorithm 4 takes the average of items in the middle  $2k$  of the  $s$  sorted buckets  $w_1/\pi_1 \leq \dots \leq w_s/\pi_s$ . This can again be done in  $O(\log s) = O(\log k + \log \log n)$  time per update using a balanced binary search tree. An alternative implementation using *biased heaps* is described Algorithm 5. The idea of the algorithm is very simple: we use heaps to keep track of  $\sum_{i \in A} w_i$ ,  $\sum_{i \in C} w_i$ ,  $\sum_{i \in A} \pi_i$ ,  $\sum_{i \in C} \pi_i$ , where  $A$  is the set of the top  $(s/2 - k)$  coordinates and  $C$  is the set of the bottom  $(s/2 - k)$  coordinates (the order is defined by  $w_i/\pi_i$ ). Using these sums together with  $\sum_{i \in [s]} w_i$  and  $\|\pi\|_1$  we can well estimate the bias.

The full streaming algorithm for  $\ell_\infty/\ell_2$  guarantee is described in Algorithm 6, which is similar to Algorithm 4 but has been augmented to fit the streaming model. The one for  $\ell_\infty/\ell_1$  guarantee can be done similarly, and we omit here.

Finally we comment on how to generate and store random hash functions in the streaming setting. In fact, we can simply choose hash functions  $g, h^i, r^i (i \in [d])$  to be 2-wise independent (and each will use  $O(1)$  space to store). This will not affect any of our analysis since we only need to use the second moment

---

**Algorithm 5: Bias-Heap**

---

**Input:**  $s$  (# of rows of CM-matrix  $\Pi$ ), and vector  $\pi$  (coordinate-wise sum of columns of matrix  $\Pi$ )

- 1 create  $s$  nodes, each of which is associated with a (key, value, id) triple  $(w_i/\pi_i, w_i, i)$  where  $w_i = 0$  and  $\pi_i$  is the  $i$ -th coordinate of  $\pi$   
/\* key is the priority of nodes in the heap \*/
- 2 set  $k \leftarrow s/4$
- 3 initialize a min-heap  $A$  for nodes 1 to  $s/2 - k - 1$
- 4 initialize a max-heap  $B$  for nodes  $s/2 - k$  to  $s$
- 5 initialize a max-heap  $C$  for nodes  $s/2 + k$  to  $s$
- 6 initialize a min-heap  $D$  for nodes 1 to  $s/2 + k - 1$
- 7  $\pi_A \triangleq \sum_{i \in \{\text{all id in } A\}} \pi_i$ ;  $\pi_C \triangleq \sum_{i \in \{\text{all id in } C\}} \pi_i$
- 8  $w_A \triangleq \sum_{i \in \{\text{all id in } A\}} w_i$ ;  $w_C \triangleq \sum_{i \in \{\text{all id in } C\}} w_i$
- 9  $w \leftarrow 0$   
/\* process updates or queries \*/
- 10 **case upon receiving an update**  $(j, \Delta)$
- 11      $w \leftarrow w + \Delta$
- 12     find node with id  $j$  in two of heaps  $A, B, C, D$  and update its  $w_j$  by adding  $\Delta$ ; update the corresponding key and maintain the heap properties
- 13     **if the key of the top node in  $A$  is smaller than that of the top of  $B$  then**
- 14         └ swap their tops and maintain heap properties
- 15     **if the key of the top node of  $C$  is larger than that of the top of  $D$  then**
- 16         └ swap their tops and maintain heap properties
- 17     update  $\pi_A, \pi_C, w_A, w_C$  if necessary
- 18 **case upon receiving a query of the bias**  $\beta$
- 19     **return**  $\frac{w - w_A - w_C}{\|\pi\|_1 - \pi_A - \pi_C}$

---

of random variables (same in the proofs for Theorem 1 and Theorem 2 for the CM-sketch and CS-sketch, see [6, 11]). Thus the total extra space to store random hash functions can be bounded by  $O(d) = O(\log n)$ , and is negligible compared with the sketch size  $O(k \log n)$ .

## 4 Experiments

### 4.1 Reference Algorithms

We use  $\ell_1$ -SKETCH/RECOVER ( $\ell_1$ -S/R) to denote Algorithm 1 and Algorithm 2 for the  $\ell_\infty/\ell_1$  guarantee, and  $\ell_2$ -SKETCH/RECOVER ( $\ell_2$ -S/R) to denote Algorithm 3 and Algorithm 4 for the  $\ell_\infty/\ell_2$  guarantee.

We use the Count-Sketch (CS) algorithm and the Count-Median (CM) algorithm as the baseline. We would also like to mention the Count-Min algorithm which was proposed in the same paper [11] as Count-Median. The Count-Min algorithm is very similar to Count-Median; they share the same sketching matrix. In fact, Count-Median can be thought as a generalization of Count-Min [11]. We experimentally compared the performance of the two algorithms on the real-world datasets we use in this paper, and found that their performances are very similar. Due to the space constraints we leave the details of the comparison in

---

**Algorithm 6:** Streaming Algorithm with  $\ell_\infty/\ell_2$  Guarantee

---

```

/*  $s = c_s k$  for a constant  $c_s \geq 4$ ;  $d = \Theta(\log n)$ ;  $g, h^1, \dots, h^d : [n] \rightarrow [s]$ ;
 $r^1, \dots, r^d : [n] \rightarrow \{-1, 1\}$  are the same as in Algorithm 4;
 $\pi \leftarrow \sum_{j \in [n]} j$ -th column of  $\Pi(g)$  */
1  $\forall i \in [d], \psi^i \leftarrow$  coordinate-wise sum of columns of the CS-matrix  $\Psi(h^i, r^i)$ 
2 initialize  $y^1, \dots, y^d$  to be all-zero vectors of length  $s$ 
3 initialize Bias-Heap in Algorithm 5 with  $s$  and  $\pi$ 
/* process updates or queries */
4 case upon an update  $(e_i, \Delta)$ 
5    $\forall t \in [d], y_{h^t(i)}^t \leftarrow y_{h^t(i)}^t + r^t(i) \cdot \Delta$ 
6   update the Bias-Heap with  $(g(e_i), \Delta)$ 
7 case upon receiving a query for computing  $x_i$ 
8   query Bias-Heap to get  $\hat{\beta}$ 
9    $z \leftarrow \text{median} \left\{ r^t(i) \cdot \left( y_{h^t(i)}^t - \psi_{h^t(i)}^t \cdot \hat{\beta} \right) \mid t \in [d] \right\}$ 
10  return  $z + \hat{\beta}$ 

```

---

Appendix A. In the main text we will only discuss Count-Median.

We also compare our algorithms with **BOMP** proposed in [28], but only on small datasets since (as we will see that) **BOMP** cannot finish in a reasonable amount of time on large datasets. We briefly describe here how **BOMP** works. To sketch a vector  $\mathbf{x} \in \mathbb{R}^n$ , **BOMP** first computes  $\mathbf{y} = \Phi \mathbf{x}$  where  $\Phi = [\phi_1, \dots, \phi_n] \in \mathbb{R}^{t \times n}$ , where each entry of  $\Phi$  is independently sampled from the Gaussian distribution  $\mathcal{N}(0, 1/t)$ . In the recovery phase **BOMP** prepends a new column  $\frac{1}{\sqrt{n}} \sum_{i=1}^n \phi_i$  to  $\Phi$  to get  $\Phi' = [\frac{1}{\sqrt{n}} \sum_{i=1}^n \phi_i, \Phi]$ , and then runs **OMP** (*Orthogonal Matching Pursuit*) on  $\mathbf{y}$  and  $\Phi'$  in  $k + 1$  iterations to recover  $\tilde{\mathbf{x}}$  as an approximation of  $\mathbf{x}$ . We used QR-decomposition to speed up the implementation of **OMP**, following the steps sketched in [25] (Section 2.3). As mentioned, in  $\ell_1$ -S/R and  $\ell_2$ -S/R, as well as CS and CM, we can decode each coordinate of  $\mathbf{x}$  efficiently and in parallel, but we cannot do this in **BOMP** since it uses **OMP** as a subroutine which is a time-expensive iterative procedure.

Finally, we also compare  $\ell_1$ -S/R and  $\ell_2$ -S/R with two simple algorithms that just use the *mean* of all coordinates in  $\mathbf{x}$  as the bias (denoted by  $\ell_1$ -mean and  $\ell_2$ -mean respectively). As mentioned earlier, using the mean of all the coordinates as the bias *cannot* give us any theoretical guarantees (in other words, it performs badly in some datasets). However, as we will see shortly, this simple heuristic works well on many real-world datasets. Thus they may be interesting to practitioners.

## 4.2 The Experiment Setup

**Datasets.** We compare the algorithms in a set of real and synthetic datasets.

- **Gaussian.** Each entry of  $\mathbf{x}$  is independently sampled from the Gaussian distribution  $\mathcal{N}(b, \sigma^2)$  where  $b$  is the *bias*. Let  $k$  be the parameter which intuitively corresponds to the number of outliers. We randomly pick  $k$  coordinates from  $\mathbf{x}$  and shift each of them by  $+1000$  or  $-1000$ .
- **Yahoo.** The Yahoo! Webscope dataset. We take the A1Benchmark/real\_24.csv dataset from ydata-labeled-time-series-anomalies-v1.0 [1]; the dimension of  $\mathbf{x}$  is  $n = 1461$ . This small dataset is mainly

used to compare  $\ell_1$ -S/R and  $\ell_2$ -S/R with BOMP, since BOMP does not scale well to larger datasets.

- **Wiki** [19]. This dataset contains pageviews to the English-language Wikipedia from 2015-03-16T00:00:00 to 2015-04-25T15:59:59. The number of pageviews of each second is recorded, we model it as a vector  $\mathbf{x}$  which has length about 3,500,000 (we added mobile views and desktop views together if they have the same timestamps).
- **WorldCup** [3]. This dataset consists of all the requests made to the 1998 World Cup Web site on May 14, 1998. We construct  $\mathbf{x}$  from those requests where each coordinate is the number of requests made in a particular second. The dimension of  $\mathbf{x}$  is therefore  $24 \times 3600 = 86,400$ .
- **Hudong** [24]. There are “related to” links between articles of the Chinese online encyclopaedia Hudong<sup>3</sup>. This dataset contains about 2,452,715 articles, and 18,854,882 edges. Each edge  $(a, b)$  indicates that in article  $a$ , there is a “related to” link pointing to article  $b$ . Such links can be added or removed by users. We consider edges as a data stream, arriving in the order of editing time. Let  $\mathbf{x}$  be the out-degree of those articles, and  $x_i$  is the number of “related to” links in article  $i$ . This dataset will be used to test our algorithms in the streaming model where we dynamically maintain a sketch for  $\mathbf{x}$ .

**Measurements.** We measure the effectiveness of the tested algorithms mainly by the tradeoffs between *sketch size* and the *recovery quality*. We also compare the *running time* of these algorithms.

For CM and  $\ell_1$ -S/R, we use  $d = 40$  copies of CM-matrices  $\Pi \in \{0, 1\}^{s \times n}$  (see Algorithm 1). The total sketch size is  $d \times s$  for CM, and  $d \times s + 20 \log n$  for  $\ell_1$ -S/R where  $20 \log n$  counts the row-dimension of the sampling matrix. Similarly, for CS and  $\ell_2$ -S/R, we use  $d = 40$  copies of CS-matrices  $\Psi \in \{0, 1\}^{s \times n}$  (see Algorithm 3). The total sketch size is  $d \times s$  for CS, and  $d \times s + s$  for  $\ell_2$ -S/R where the second term  $s$  counts the additional CM-matrix we use for estimating the bias. We will vary  $s$  to get multiple sketch-size versus accuracy tradeoffs.

Being consistent with our theorems (Theorem 4, Corollary 2, Theorem 3, Corollary 1), we will use the following two measurements. Recall that  $\hat{\mathbf{x}}$  is the approximation of  $\mathbf{x}$  given by the recover scheme.

- Average error:  $\|\mathbf{x} - \hat{\mathbf{x}}\|_1 / n$ .
- Maximum error:  $\|\mathbf{x} - \hat{\mathbf{x}}\|_\infty$ .

As mentioned earlier, BOMP only tries to recover the  $k$  coordinates that deviate the most from the bias. Let  $\hat{O}_k$  be the set of indices of the  $k$  outlier coordinates that BOMP recovers. We thus use  $\sum_{i \in \hat{O}_k} |x_i - \hat{x}_i| / k$  and  $\max_{i \in \hat{O}_k} |x_i - \hat{x}_i|$  (which is at most  $\|\mathbf{x} - \hat{\mathbf{x}}\|_\infty$ ) as estimations of the average error and the maximum error of BOMP, respectively.

In the experiments for outlier detection, we use  $\ell_1$ -error to measure the performance of the algorithms: Let  $m$  be the median of  $\mathbf{x}$ ,  $O_k$  the set of the indices of the  $k$  outliers in  $\mathbf{x}$  (those deviate from  $m$  the most) and  $\hat{O}_k$  the set of the indices of the outliers recovered by the algorithm. The  $\ell_1$ -error is defined as

$$1 - \frac{\sum_{i \in \hat{O}_k} |x_i - m|}{\sum_{i \in O_k} |x_i - m|}.$$

It is clear that if  $\hat{O}_k$  is equal to  $O_k$ , then  $\ell_1$ -error is zero. Moreover,  $\ell_1$ -error is very robust to the choice of  $k$ . In contrast, the precision and recall are very sensitive to the choice of  $k$ . To see this, suppose that

---

<sup>3</sup><http://www.hudong.com/>

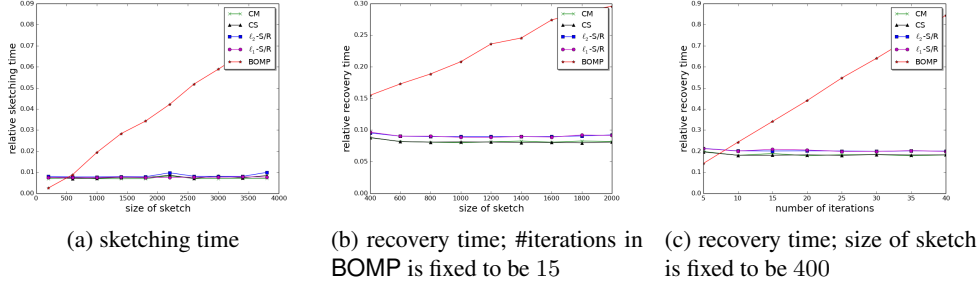


Figure 1: Comparison of sketching and recovery time. Gaussian dataset with  $n = 10,000$

there are two equally large outliers in  $\mathbf{x}$  and we set  $k = 1$ , even with a perfect recovery we may end up with  $|O_k \cap \hat{O}_k| = 0$ .

We repeat each experiment 10 times and then take the average of the results.

**Computation Environments.** All algorithms were implemented in python 2.7 + numpy + scipy. All experiments were run in a server with 32GB RAM and an Intel Xeon E5-2650 v2 8-core processor; the operating system is Red Hat Enterprise Linux 6.7. Multi-thread executions are used in all of our experiments except when comparing our algorithms with BOMP (Section 4.3).

### 4.3 Running Time

We compare the running time of the five algorithms (BOMP, CM, CS,  $\ell_1$ -S/R,  $\ell_2$ -S/R). Since BOMP is inherent sequential, to be fair we only use single-thread executions in this comparison.

We present our experimental results in Figure 1 where the size of the vector is set to be  $n = 10,000$ . In Figure 1a, we fix the size of the vector  $\mathbf{x}$  and vary the size of the sketch. All of CS, CM,  $\ell_1$ -S/R and  $\ell_2$ -S/R have sketching time independent of the sketch size, while BOMP's sketching time grows linearly; when sketch size is set to be 3000, BOMP's running time is 7+ times larger than the others.

In Figure 1b and Figure 1c we compare the recovery time. In Figure 1b we fix the number of iterations for BOMP and vary the size of the sketch. We have observed that BOMP's recovery time grows linearly while others' are independent of the sketch size. In Figure 1c we fix the size of the sketch and vary the number of iterations for BOMP. Again BOMP's recovery time grows linearly with respect to the number of iterations. When we set the number of iterations to be 40, BOMP's running time is 4+ times larger than other four.

We next increase the size of the vector  $n$  to 1,000,000, set the number of iteration  $k$  to be 1000, and set the sketch size to be 10,000. We have observed that all of CS, CM,  $\ell_1$ -S/R and  $\ell_2$ -S/R return  $\hat{\mathbf{x}}$  within 10 minutes, while BOMP *cannot* return any result within 5 hours.

Our experiments confirm with the theoretical guarantees. Theoretically, CM, CS,  $\ell_1$ -S/R and  $\ell_2$ -S/R only need  $O(n \log n)$  time to sketch a vector  $\mathbf{x} \in \mathbb{R}^n$ , while BOMP needs  $O(s \cdot n)$  time, where  $s$  is the sketch size which can be significantly larger than  $\log n$ . The reason is that both CS-Matrix and CM-Matrix are sparse matrices with only one non-zero entry in each column, while BOMP uses a dense  $s \times n$  random Gaussian matrix. In the recovery step, CM, CS,  $\ell_1$ -S/R and  $\ell_2$ -S/R only need  $O(n \log n)$  time to recover all coordinates, while BOMP needs  $O(s \cdot n \cdot k + sk^2)$  to recover just the top- $k$  coordinates, even when the QR-Factorization speed-up (see [26] section 2.3 QR-1) is used. The  $sk$  multiplicative overhead becomes significant, often prohibitive, when the size of the vector  $n$  is large and we need a good accuracy guarantee.

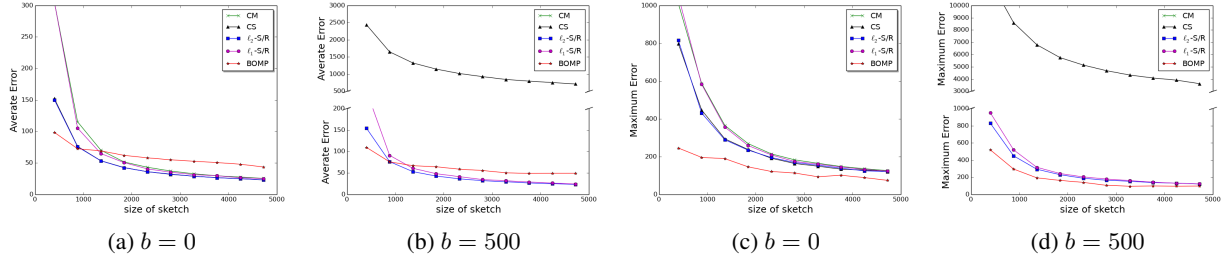


Figure 2: Gaussian dataset;  $n = 10,000$ ; In BOMP the #iterations is 10. Errors for CM are too large to present in Figure 2b, 2d

#### 4.4 Recovery Accuracy

We next compare the accuracies of tested algorithms.

**Gaussian dataset with  $n = 10,000$ .** In Figure 2 we run experiments on Gaussian dataset with  $n = 10,000$ . We vary the bias  $b$  and the sketch size. Figure 2a and 2c show the average error and the maximum error of five algorithms when setting  $b = 0$  (that is, no bias). We can see all algorithms have similar errors (within a constant factor of 2). Note in Figure 2c and Figure 2d, BOMP seems to have slightly better recovery quality than CS, this is because the maximum error defined for BOMP is an underestimation of the maximum error over all coordinates. See the error definitions in the “measurements” paragraph.

In Figure 2b and Figure 2d we set the bias  $b = 500$ . In this case,  $\ell_1$ -S/R,  $\ell_2$ -S/R and BOMP have similar recovery qualities, and are significantly better than CS and CM. The errors of CM and CS are 10+ times larger than BOMP,  $\ell_1$ -S/R and  $\ell_2$ -S/R.

**Gaussian dataset with  $n = 5,000,000$ .** Figure 3 and Figure 4 show the average and maximum errors of  $\ell_1$ -S/R,  $\ell_2$ -S/R, CM and CS respectively on a much larger Gaussian dataset. We vary the bias  $b$ , the number of “outliers”  $k$ , and the sketch size.

First note that as long as the sketch size is larger than  $k \log n$ , the change of  $k$  does not affect the recovery quality by much. This is in fact predicted from the tail-form error (i.e.  $\text{Err}_p^k$ ) of CM, CS,  $\ell_1$ -S/R and  $\ell_2$ -S/R, since the rest of the  $n - k$  coordinates only have Gaussian noise.

When  $b$  varies, our conclusion is similar to that on the smaller Gaussian dataset (Figure 2), that is, all algorithms have similar recovery qualities when  $b = 0$ ; and when  $b = 500$ ,  $\ell_1$ -S/R and  $\ell_2$ -S/R have comparable performance and significantly outperform CM and CS.

**Yahoo dataset.** Figure 5 shows the experimental results for Yahoo dataset. We vary the sketch size and fix the number of iterations to be 10 in BOMP. It is clear from Figure 5 that BOMP and  $\ell_2$ -S/R have best recovery qualities, and  $\ell_1$ -S/R is slightly worse – it has errors 2 – 10 times larger. Compared with  $\ell_2$ -S/R and BOMP, CS has errors that are 30+ times larger and CM has errors that are 100+ times larger.

We next move to the datasets with much larger sizes. As mentioned earlier, BOMP cannot scale to these datasets in our computational environments. We thus only compare  $\ell_1$ -S/R and  $\ell_2$ -S/R with CM and CS.

**Wiki dataset.** Figure 6 shows the accuracies of  $\ell_1$ -S/R,  $\ell_2$ -S/R, CM and CS on Wiki dataset. We have observed that when varying the sketch size,  $\ell_2$ -S/R always achieves the best recovery quality. For example, when sketch size is 200,000, the average error of CS is 10+ times larger than  $\ell_2$ -S/R,  $\ell_1$ -S/R is 20+ times larger and CM is 100+ times larger.

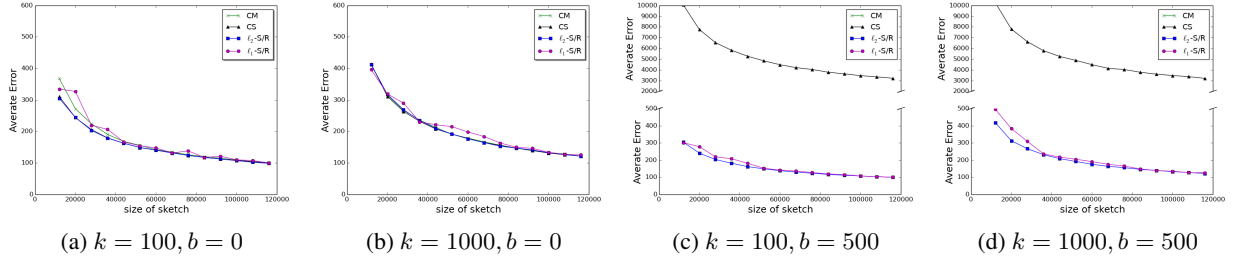


Figure 3: Gaussian dataset; error measured by average error.  $n = 5,000,000$ .  $b, k$  varies and  $\sigma = 15$ . Errors for CM are too large to present in Figure 3c, 3d

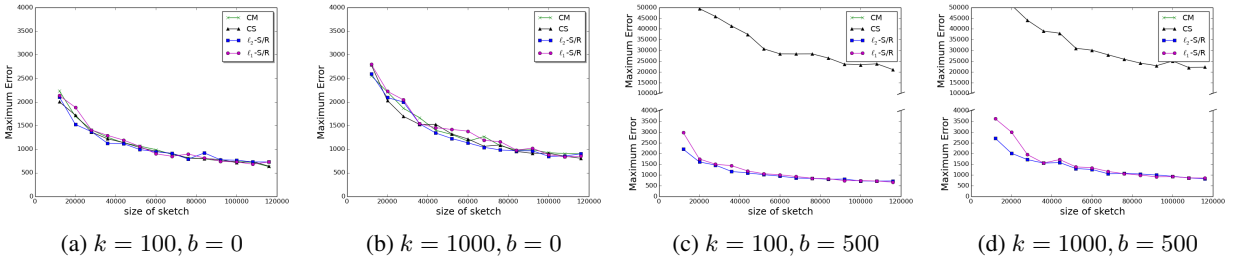


Figure 4: Gaussian dataset; error measured by maximum error.  $n = 5,000,000$ .  $b, k$  varies and  $\sigma = 15$ . Errors for CM are too large to present in Figure 4c, 4d

It is interesting to observe that CS has better recovery quality than  $\ell_1$ -S/R on Wiki dataset, which is different from the result we get on Gaussian dataset (Figure 3 and Figure 4). An explanation for this phenomenon is that in CS we use random signs  $+1, -1$  to reduce/cancel the noise (contributed by colliding coordinates) in each hashing bucket, while in  $\ell_1$ -S/R we do not. In Gaussian the “perturbation” of each  $x_i$  around the bias is symmetric so both algorithms achieve good cancellations. But in Wiki the perturbation is not symmetric so the cancellation in  $\ell_1$ -S/R becomes less great (even after the de-bias, but de-bias makes  $\ell_1$ -S/R much better than CM); while the cancellation in CS still works well due to the random  $+1, -1$  signs (even without de-bias, but due to the lack of de-bias CS is worse than  $\ell_2$ -S/R).

#### 4.5 Application to Outlier Detection

On datasets Gaussian, Yahoo and WorldCup we compare our algorithms with CS and CM in terms of outlier detection. In all algorithms, we pick the top- $k$  coordinates of  $\hat{\mathbf{x}}$  that deviate the most from the estimated bias (0 in the case of CS and CM).

Figure 7 shows the  $\ell_1$ -error of different algorithms. It can be seen that  $\ell_1$ -S/R and  $\ell_2$ -S/R significantly outperform CS and CM in terms of  $\ell_1$ -error.

#### 4.6 Comparisons with $\ell_1$ -mean and $\ell_2$ -mean

We compare  $\ell_1$ -S/R and  $\ell_2$ -S/R with  $\ell_1$ -mean and  $\ell_2$ -mean in Figure 8. From Figure 8a, 8b and 8c we can see that when the shift of outliers increases, the recovery quality of  $\ell_2$ -mean and  $\ell_1$ -mean decreases. This

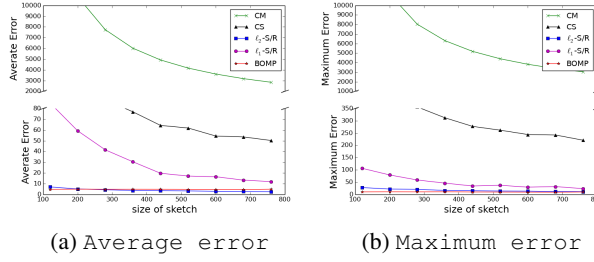


Figure 5: Yahoo dataset;  $n = 1461$ . In BOMP the #iterations is 10

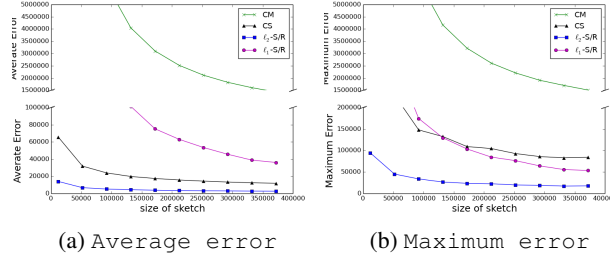


Figure 6: Wiki dataset;  $n = 3,513,600$

is because the mean of the vector goes further away from the bias when the outliers deviate more from the median. On the other hand, it can be observed that both  $\ell_1$ -S/R and  $\ell_2$ -S/R are very robust to the change of shifts.

In Figure 8d and 8e, however,  $\ell_2$ -S/R,  $\ell_1$ -mean and  $\ell_2$ -mean give very close results. This indicates that for real-world datasets, the mean of the vector is possibly a good estimation of the bias as well.

## 4.7 Distributed and Streaming Implementations

As mentioned in the introduction, it is straightforward to implement our bias-aware sketches in the distributed model by making use of the linearity. Moreover, their performance in the distributed model can be *fully* predicted by the centralized counterparts – the total communication will just be the number of sites times the size of the sketch, and the time costs at the sites and the coordinator will be equal to the sketching time and recovery time respectively.<sup>4</sup> Therefore, our experiments in the centralized model can also speak for that in the distributed model.

We implemented our bias-aware sketches in the streaming model; the algorithms are presented in Section 3.4. We have run our algorithms on the streaming dataset Hudong. We update the sketch in the online fashion, and recover the entire  $\hat{\mathbf{x}}$  after feeding in the whole dataset. Figure 9a and Figure 9b show that the recovery error of CS is 3+ times larger than  $\ell_2$ -S/R. Both  $\ell_1$ -S/R and CM are worse than CS and  $\ell_2$ -S/R, but  $\ell_1$ -S/R is consistently better than CM. All of these four algorithms have similar processing time per update (within a constant factor of 2); see Figure 9c.

<sup>4</sup>Regarding the random hash functions, the coordinator can simply generate  $g, h^1, \dots, h^d : [n] \rightarrow [s]; r^1, \dots, r^d : [n] \rightarrow \{-1, 1\}$  at the beginning and send to each site, which only incurs an extra of  $O(\log n)$  communication on each channel and is thus negligible.



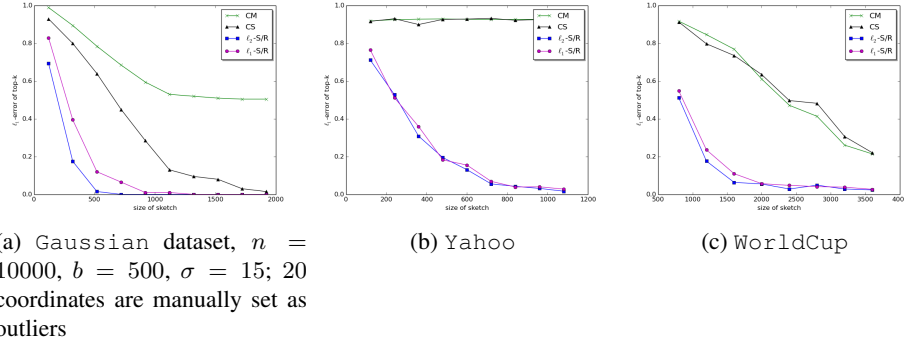
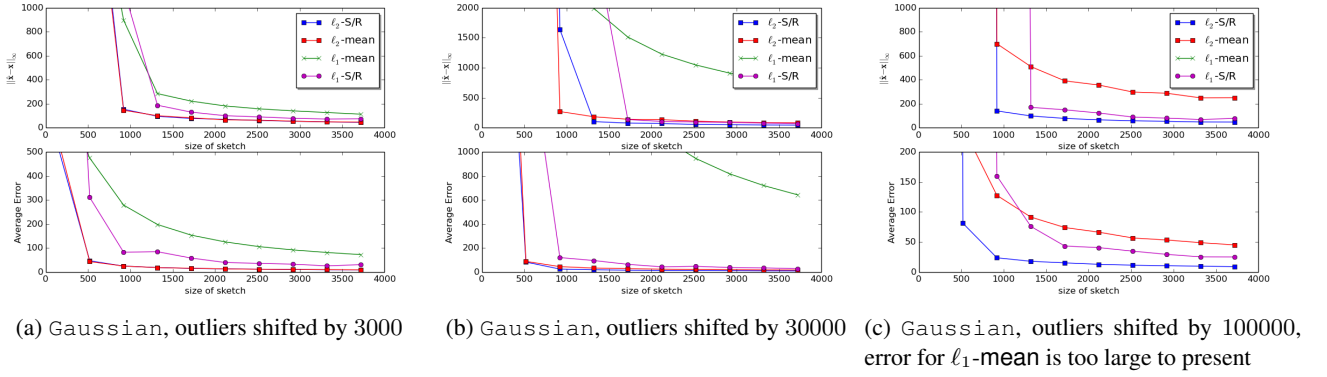
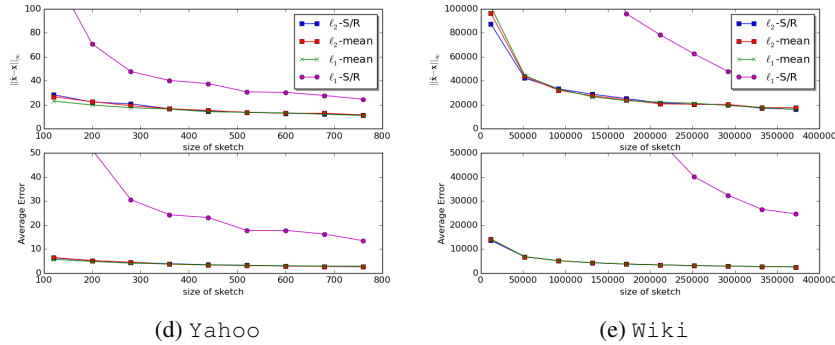


Figure 7: Outlier detection,  $k = 10$  in all cases



Gaussian dataset,  $n = 10,000$ , bias  $b = 500$ ,  $\sigma = 15$ ; there are 20 outliers



Yahoo, Wiki datasets;  $\ell_2$ -S/R,  $\ell_2$ -mean and  $\ell_1$ -mean overlap with each other

Figure 8: Comparisons with algorithms estimating the bias by the mean

## 4.8 Summary of Experimental Results

We now summarize our experimental results. We have observed that in terms of recovery quality,  $\ell_1$ -S/R strictly outperforms CM, and  $\ell_2$ -S/R strictly outperforms CS. In general  $\ell_2$ -S/R is better than  $\ell_1$ -S/R,

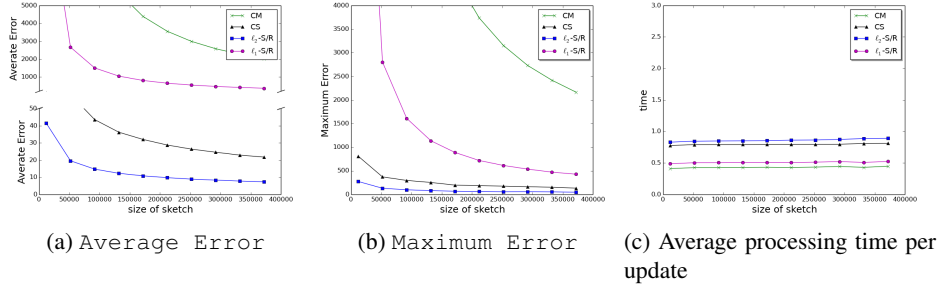


Figure 9: Hudong dataset;  $n = 2,232,285$ .

especially when the noise around the bias is not symmetric.  $\ell_2$ -S/R and BOMP have comparable recovery quality, however  $\ell_2$ -S/R has many advantages over BOMP, including (1)  $\ell_2$ -S/R is significantly faster than BOMP in both sketching and recovery phases, and can be easily parallelized; and (2)  $\ell_2$ -S/R can be used to efficiently answer point queries in the streaming model, while BOMP can not. We also note that  $\ell_2$ -S/R and  $\ell_1$ -S/R have rigorous theoretical guarantees while BOMP does not.

Our experiments also show that the mean of the vector turns out to be a fairly good estimation of the bias for real-world datasets (though it can be arbitrarily bad for adversarial inputs). This observation can help to simplify the implementation of  $\ell_1$ -S/R and  $\ell_2$ -S/R in some situations.

## References

- [1] Yahoo! Webscope dataset, ydata-labeled-time-series-anomalies-v1.0. [http://labs.yahoo.com/Academic\\_Relations](http://labs.yahoo.com/Academic_Relations).
- [2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29. ACM, 1996.
- [3] M. Arlitt and T. Jin. World cup web site access logs, august 1998. URL <http://ita.ee.lbl.gov/html/contrib/WorldCup.html>, 1998.
- [4] R. Baraniuk, M. A. Davenport, M. F. Duarte, and C. Hegde. An introduction to compressive sensing. *Connexions e-textbook*, 2011.
- [5] E. J. Candès, J. K. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.
- [6] M. Charikar, K. C. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703, 2002.
- [7] G. Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*, 2011.
- [8] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24. VLDB Endowment, 2005.

- [9] G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *VLDB J.*, 19(1):3–20, 2010.
- [10] G. Cormode, T. Johnson, F. Korn, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Holistic udafs at streaming speeds. In *SIGMOD*, pages 35–46. ACM, 2004.
- [11] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.
- [12] G. Cormode and S. Muthukrishnan. Combinatorial algorithms for compressed sensing. In *SIROCCO*, pages 280–294, 2006.
- [13] F. Deng and D. Rafiei. New estimation algorithms for streaming data: Count-min can do more, 2007.
- [14] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.
- [15] D. L. Donoho, M. Elad, and V. N. Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Trans. Information Theory*, 52(1):6–18, 2006.
- [16] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.
- [17] A. C. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.
- [18] A. C. Gilbert, S. Muthukrishnan, and M. Strauss. Approximation of functions over redundant dictionaries using coherence. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 243–252, 2003.
- [19] O. Keyes. Wiki-Pageviews, english wikipedia pageviews by second. <http://datahub.io/dataset/english-wikipedia-pageviews-by-second>, April, 2015.
- [20] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. *Communications of the ACM*, 54(6):114–123, 2011.
- [21] G. T. Minton and E. Price. Improved concentration bounds for count-sketch. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 669–686, 2014.
- [22] R. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.
- [23] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.
- [24] X. Niu, X. Sun, H. Wang, S. Rong, G. Qi, and Y. Yu. Zhishi.me – weaving Chinese linking open data. In *Proc. Int. Semantic Web Conf.*, pages 205–220, 2011.
- [25] B. L. Sturm and M. G. Christensen. Comparison of orthogonal matching pursuit implementations. In *EUSIPCO*, pages 220–224, 2012.

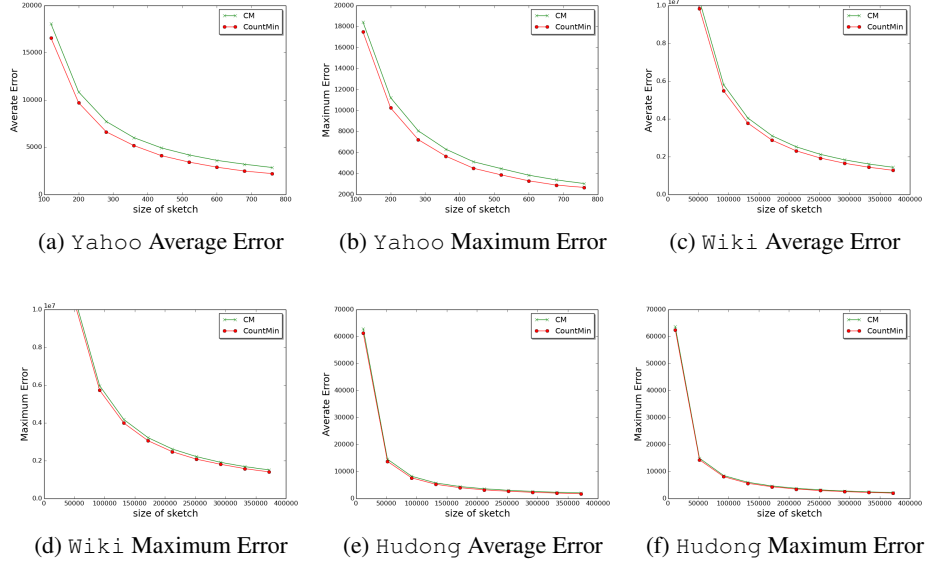


Figure 10: Count-Median versus Count-Min in Yahoo, Wiki and Hudong

- [26] B. L. Sturm and M. G. Christensen. Comparison of orthogonal matching pursuit implementations. In *EUSIPCO*, pages 220–224, 2012.
- [27] D. Van Gucht, R. Williams, D. P. Woodruff, and Q. Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *PODS*, pages 199–212. ACM, 2015.
- [28] Y. Yan, J. Zhang, B. Huang, X. Sun, J. Mu, Z. Zhang, and T. Moscibroda. Distributed outlier detection using compressive sensing. In *SIGMOD*, pages 3–16. ACM, 2015.

## A A Comparison of Count-Median and Count-Min

The only difference between Count-Median and Count-Min is that in Count-Min,  $\hat{x}_j$  is set to be  $\min_{i \in [d]} (\Pi(h^i) \mathbf{x})_{h^i(j)}$  instead of median (compared with Theorem 1). The theoretical error guarantees of the two algorithms are asymptotically the same, but Count-Min can only be used to handle non-negative vectors  $\mathbf{x}$ . As indicated by [11], Count-Median can be thought as a generalization of Count-Min.

Figure 10 gives a comparison between the Count-Median algorithm and the Count-Min algorithm, on datasets Yahoo, Wiki and Hudong. It can be seen that their performance in terms of average and maximum recovery errors is very close.